Secure Two-Party Computation with Reusable Bit-Commitments, via a Cut-and-Choose with Forge-and-Lose Technique*

(Extended abstract – September 11, 2013)

Luís T. A. N. Brandão†

University of Lisbon
Faculty of Sciences / LaSIGE
Lisboa, Portugal
lbrandao@fc.ul.pt

Carnegie Mellon University
Electrical & Computer Engineering
Pittsburgh, USA
lbrandao@cmu.edu

Abstract. A *secure two-party computation* (S2PC) protocol allows two parties to compute over their combined private inputs, as if intermediated by a trusted third party. In the malicious model, this can be achieved with a *cut-and-choose* of *garbled circuits* (C&C-GCs), where some GCs are *verified* for correctness and the remaining are *evaluated* to determine the circuit output. This paper presents a new C&C-GCs-based S2PC protocol, with significant advantages in efficiency and applicability. First, in contrast with prior protocols that require a majority of *evaluated* GCs to be correct, the new protocol only requires that at least one *evaluated* GC is correct. In practice this reduces the total number of GCs to approximately one third, for the same statistical security goal. This is accomplished by augmenting the C&C with a new *forge-and-lose* technique based on bit commitments with trapdoor. Second, the output of the new protocol includes reusable XOR-homomorphic bit commitments of all circuit input and output bits, thereby enabling efficient linkage of several S2PCs in a reactive manner. The protocol has additional interesting characteristics (which may allow new comparison tradeoffs), such as needing a low number of exponentiations, using a 2-out-of-1 type of oblivious transfer, and using the C&C structure to statistically verify the consistency of input wire keys.

Keywords: secure two-party computation, cut-and-choose, garbled circuits, forge-and-lose, homomorphic bit-commitments with trapdoor.

1 Introduction

Secure two-party computation is a general cryptographic functionality that allows two parties to interact as if intermediated by a *trusted third party* [Gol04]. A canonical example is *the millionaire's problem* [Yao82], where two parties

* © IACR 2013. This article is the final version submitted by the author to the IACR and to Springer-Verlag on September 11, 2013. A similar version will be published by Springer-Verlag. The full version is available at the Cryptology ePrint Archive [Bra13].

† Support for this research was provided by the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program under Grant SFRH/BD/33770/2009, and through the Multiannual Funding Programme (LASIGE), while the author was a Ph.D. student at FCUL-DI-LaSIGE and CMU-ECE.

find who is the richer of the two, without revealing to the other any additional information about the amounts they own. Applications of secure computation can be envisioned in many cases where mutually distrustful parties can benefit from learning something from their combined data, without sharing their inputs [Kol09]. For example, two parties may evaluate a data mining algorithm over their combined databases, in a privacy-preserving manner [LP02]. On a different example, one party with a private message may obtain a respective message authentication code calculated with a secret key from another party (i.e., a blind MAC) [PSSW09]. This paper considers secure two-party evaluation of Boolean circuits, henceforth denoted “S2PC”, which can be used to solve the mentioned examples. Each party begins the interaction with a private input encoded as a bit-string, and a public specification of a Boolean circuit that computes an intended function. Then, the two parties interact so that each party learns only the output of the respective circuit evaluated over both private inputs. Probabilistic functionalities can be implemented by letting the two parties hold additional random bits as part of their inputs.

This paper focuses on the *malicious model*, where parties might maliciously deviate from the protocol specification in a computationally bounded way. Furthermore, within the *standard model* of cryptography, adopted herein, it is assumed that some problems are computationally intractable, such as those related with inverting trapdoor permutations. Security is defined within the ideal/real simulation paradigm [Can00]; i.e., a protocol is said to implement S2PC if it *emulates* an ideal functionality where a trusted third party mediates the communication and computation between the two parties. The trusted party receives the private inputs from both parties, makes the intended computation locally and then delivers the final private outputs to the respective parties.

As a starting point, this paper considers the *cut-and-choose* (C&C) of *garbled circuits* (GCs) approach to achieve S2PC. Here, a circuit constructor party ($\mathrm{P_A}$) builds several GCs (cryptographic versions of the Boolean circuit that computes the intended function), and then the other party, the circuit evaluator ($\mathrm{P_B}$), verifies some GCs for correctness and evaluates the remaining to obtain the information necessary to finally decide a correct circuit output. Recently, this approach has had the best reported efficiency benchmark [KSS12; FN13] for S2PC protocols with a constant number of rounds of communication.

1.1 Contributions. This paper introduces a new *bit commitment* (BitCom) approach and a new evaluation technique, dubbed *forge-and-lose*, and blends them into a C&C approach, to achieve a new C&C-GCs-based S2PC protocol with significant improvements in applicability and efficiency.

Applicability. The new protocol achieves S2PC-with-BitComs, as illustrated in Fig. 1. Specifically, both parties receive random BitComs of all circuit input and output bits, with each party also learning the decommitments of only her respective circuit input and output bits. This is an augmented version of secure circuit evaluation. Given the reusability of BitComs, the protocol can be taken as a building block to achieve other goals, such as reactive linkage of several S2PCs, efficiently and securely linking the input and output bits of one execution with the input bits of subsequent executions. Furthermore, given the

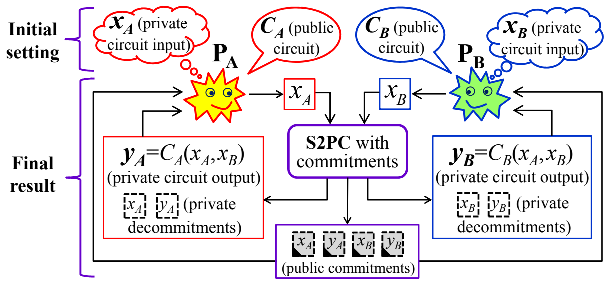
Initial setting
x_A (private circuit input)
C_A (public circuit)
C_B (public circuit)
x_B (private circuit input)
P$_A$
P$_B$
x_A
x_B
Final result
$y_A=C_A(x_A,x_B)$ (private circuit output)
x_A y_A (private decommitments)
S2PC with commitments
$y_B=C_B(x_A,x_B)$ (private circuit output)
x_B y_B (private decommitments)
x_A y_A x_B y_B (public commitments)

Fig. 1: **Secure Two-Party Computation with Committed Inputs and Outputs.** Legend: P$_A$ and P$_B$ (the names of the two parties); x_p, y_p and C_p (the private circuit input, the private circuit output and the public circuit specification of party P$_p$, respectively, with p being A or B); ▨ and ☐ (commitment and decommitment, respectively, of the variable inscribed inside the dashed square). Colors red, blue and purple are related with P$_A$, P$_B$ and both parties, respectively.

XOR-homomorphic properties of these BitComs, a party may use efficient specialized *zero-knowledge proofs* (ZKPs) to prove that her private input bits in one execution satisfy certain non-deterministic polynomially verifiable relations with the private input and output bits of previous executions.[1] In previous C&C-GCs-based solutions, without committed inputs and outputs with homomorphic properties, such general linkage would be conceivable but using more expensive ZKPs of correct behavior.

The main technical description in this paper is focused on a standalone 1-output protocol, where the two parties, P$_A$ and P$_B$, interact so that only P$_B$ learns a circuit output.[2] In the new BitCom approach, the two possible decommitments of the BitCom of each circuit input or output bit (independent of the number of GCs) are connected to the two keys of the respective input or output wire of each GC, via a new construction dubbed *connector*. P$_A$ commits to these *connectors* and then reveals them partially for *verification* or *evaluation*. This ensures, within the C&C, the correctness of circuit input keys and the privacy of decommitments of BitComs, without requiring additional ZKPs. The BitCom approach enables particularly efficient extensions of this 1-output protocol into 2-output protocols where both parties learn a respective private circuit-output.

Efficiency. The new protocol requires only an optimal minimum number of GCs in the C&C, for a certain soundness guarantee (i.e., for an upper bound on the probability with which a malicious P$_A$ can make P$_B$ accept an incorrect output). Specifically, by only requiring that at least one evaluation GC is correct, the total number of GCs is reduced asymptotically about 3.1 times, in comparison with the previously best known C&C-GCs configuration [SS11] that required a correct majority of evaluation GCs. The significance of this im-

[1] For simplicity, "ZKPs" is used hereafter both for ZK *proofs* and for ZK *arguments.*

[2] The "1-output" characterization refers to only one party learning a circuit output, though in rigor the protocol implements a probabilistic 2-output functionality (as both parties receive random BitComs).

provement stems from the number of GCs being the source of most significant cost of C&C-GCs-based S2PC protocols, for circuits of practical size. **Remark:** two different techniques [Lin13; HKE13] developed in concurrent research also just require a single evaluation GC to be correct – a brief comparison is made in §7.1, but the remaining introductory part of this paper only discusses the typical C&C-GCs approach that requires a correct majority of evaluation GCs.

The reduction in number of GCs is achieved via a new forge-and-lose technique, providing a path by which $\mathrm{P_B}$ can recover the correct final output when there are inconsistent outputs in the evaluated GCs. Assume that $\mathrm{P_A}$ is able to *forge* a GC; i.e., build an incorrect GC that, if selected for *evaluation*, degarbles smoothly into an output that cannot be perceived as incorrect. Then, $\mathrm{P_B}$ somehow combines the forged output with a correct output, in a way that reveals a secret key (a trapdoor) with which the input of $\mathrm{P_A}$ has previously been encrypted (committed). In this way, $\mathrm{P_A}$ *loses* privacy of her input bits, enabling $\mathrm{P_B}$ to compute the intended circuit output in the clear.

The protocol can be easily adjusted to integrate several optimizations in communication and memory, such as *random seed checking* [GMS08] and *pipelining* [HEKM11]. Since the garbling scheme is abstracted, the protocol is also compatible with many garbling optimizations, e.g., *point and permute* [NPS99], *XOR for free* [KS08b], *garbled row reduction* [PSSW09], *dual-key cipher* [BHR12].

1.2 Roadmap. The remainder of this paper is organized as follows. Section 2 reviews the basic building blocks of the typical C&C-GCs approach and some properties of BitCom schemes. Section 3 introduces a new BitCom approach, explaining how BitComs can be *connected* to circuit input and output wire keys, to ensure the consistency of the keys across different GCs. Section 4 describes the forge-and-lose technique, achieving a major efficiency improvement over the typical cut-and-choose approach. Section 5 presents the new protocol for 1-output S2PC-with-BitComs, where only one party learns a private circuit-output, and both parties learn BitComs of the input and output bits of both parties. Section 6 comments on the complexity of the protocol and shows how the BitCom approach enables efficient linkage of S2PCs. Section 7 compares some aspects of related work. The full version of this paper [Bra13] includes a more formal description, analysis and optimization of the protocol and a proof of security.

2 Background

2.1 C&C-GCs-based S2PC

Basic garbled-circuit approach. The theoretical feasibility of S2PC, for functions efficiently representable by Boolean circuits, was initially shown by Yao [Yao86].[3] In the *semi-honest model* (where parties behave correctly during the protocol) simplified to the 1-output setting, only one of the parties ($\mathrm{P_B}$) intends to learn the output of an agreed Boolean circuit that computes the desired function. The *basic GC approach* starts with the other party ($\mathrm{P_A}$) building a GC – a

[3] See [BHR12, §1] for a brief historical account of the origin of the garbled-circuit approach, including references to [GMW87; BMR90; NPS99].

cryptographic version of the Boolean circuit, which evaluates keys (e.g., random bit-strings) instead of clear bits. The GC is a directed acyclic graph of garbled gates, each receiving keys as input and outputting new keys. Each gate output key has a corresponding underlying bit (the result of applying the Boolean gate operation to the bits underlying the corresponding input keys), but the bit correspondence is hidden from $\mathrm{P_B}$. $\mathrm{P_A}$ sends the GC and one circuit input key per each input wire to $\mathrm{P_B}$. Then, $\mathrm{P_B}$ obliviously evaluates the GC, learning only one key per intermediate wire but not the respective underlying bit. Finally, each circuit output bit is revealed by a special association with the key learned for the respective circuit output wire. Lindell and Pinkas [LP09] prove the security of a version of Yao's protocol (valid for a 2-output setting).

There are many known proposals for garbling schemes [BHR12]. This paper abstracts from specific constructions, except for making the typical assumptions that: (i) with two valid keys per circuit input wire (and possibly some additional randomness used to generate the GC), $\mathrm{P_B}$ can *verify the correctness* of the GC, in association with the intended Boolean circuit, and determine the bit underlying each input and output key; and (ii) with a single key per circuit input wire, $\mathrm{P_B}$ can *evaluate* the GC, learning the bits corresponding to the obtained circuit output keys, but not learn additional information about the bit underlying the single key obtained for each input wire of $\mathrm{P_A}$ and for each intermediate wire.

Oblivious transfer. An essential step of the basic GC-based protocol requires, for each circuit input wire of $\mathrm{P_B}$ (the GC-evaluator), that $\mathrm{P_A}$ (the GC-constructor) sends to $\mathrm{P_B}$ the key corresponding to the respective input bit of $\mathrm{P_B}$, but without $\mathrm{P_A}$ learning what is the bit value. This is typically achieved with 1-out-of-2 *oblivious transfer*s (OTs) [Rab81; EGL85; NP01], where the sender ($\mathrm{P_A}$) selects two keys per wire, but the receiver ($\mathrm{P_B}$) only learns one of its choice, without the sender learning which one. Some protocols use enhanced variations, e.g., committing OT [CGT95], committed OT [KS06], cut-and-choose OT [LP11], authenticated OT [NNOB12], string-selection OT [KK12]. In practice, the computational cost of OTs is often significant in the overall complexity of protocols, though asymptotically the cost can be amortized with techniques that allow extending a few OTs to a large number of them [Bea96; IKNP03; NNOB12].

The new protocol presented in this paper uses OTs at the BitCom level, to coordinate decommitments between the two parties, as follows. For each circuit input bit of $\mathrm{P_B}$, $\mathrm{P_B}$ selects a bit encoding (a decommitment) and uses it to produce the respective BitCom. Then, $\mathrm{P_A}$ uses a trapdoor to learn *two* decommitments (i.e., bit-encodings for the two bits) for the same BitCom. These OTs are herein dubbed *2-out-of-1 OTs*, since one party chooses *one* value and leads the other party to learn *two* values. This is in contrast with the typical 1-out-of-2 OT (commonly used directly at the level of wire keys), where $\mathrm{P_A}$ chooses *two* keys and leads $\mathrm{P_B}$ to learn *one* of them.

Cut-and-choose approach. Yao's protocol is insecure in the malicious model. For example, a malicious $\mathrm{P_A}$ could construct an undetectably incorrect GC, by changing the Boolean operations underlying the garbled gates, but maintaining the correct graph topology of gates and wires. To solve this, Pinkas [Pin03] proposed a C&C approach, achieving 2-output S2PC via a single-path approach where only $\mathrm{P_B}$ evaluates GCs. A simplified high level description follows. $\mathrm{P_A}$ constructs a set of GCs. $\mathrm{P_B}$ *cuts* the set into two complementary subsets and *chooses* one to *verify* the correctness of the respective GCs. If no problem is found, $\mathrm{P_B}$ *evaluates* the remaining GCs to obtain, from a consistent majority, its own output bits and a masked version of the output of $\mathrm{P_A}$. $\mathrm{P_B}$ sends to $\mathrm{P_A}$ a modified version of the masked output of $\mathrm{P_A}$, without revealing from which GC it was obtained. Finally, $\mathrm{P_A}$ unmasks her final output bits. This approach has two main inherent challenges: (1) how to *ensure that input wire keys are consistent across GCs, such that equivalent input wires receive keys associated with the same input bits (in at least a majority of evaluated GCs)*; (2) how to *guarantee that the modified masked-output of P_A is correct and does not leak private information of P_B*. Progressive solutions proposed across recent years have solved subtle security issues, e.g., the selective-failure-attack [MF06; KS06], and improved the practical efficiency of C&C-GC-based methods [LP07; Woo07; KS08a; NO09; PSSW09; LP11; SS11]. As a third challenge, the number of GCs still remains a primary source of inefficiency, in these solutions that require a correct majority of GCs selected for evaluation. For example, achieving 40 bits of statistical security[4] requires at least 123 GCs (74 of which are for *verification*). Asymptotically, the optimal C&C partition (three fifths of verification GCs) leads to about 0.322 bits of statistical security per GC [SS11].

The BitCom approach developed in this paper deals with all these challenges. First, taking advantage of XOR-homomorphic BitComs, the verification of consistency of input wire keys of both parties is embedded in the C&C, without an ad-hoc ZKP of consistency of keys across different GCs. Second, $\mathrm{P_B}$ can directly learn, from the GC evaluation, decommitments of BitComs of one-time-padded (i.e., masked) output bits of $\mathrm{P_A}$, and then simply send these decommitments to $\mathrm{P_A}$. Privacy is preserved because the decommitments do not vary with the GC index. Correctness is ensured because the decommitments are verifiable (i.e., authenticated) against the respective BitComs. The BitCom approach also enables achieving 2-output S2PC via a dual-path execution approach – the parties play two 1-output S2PCs, with each party playing once as GC evaluator of only her own intended circuit, using the same BitComs of input bits in both executions.[5] Third, the BitCom approach enables the forge-and-lose technique, which reduces the correctness requirement to only having at least one correct *evaluation* GC, thus increasing the statistical security to about 1 bit per GC.

[4] The number of bits of statistical security is the additive inverse of the logarithm base 2 of the maximum error probability, i.e., for which a malicious $\mathrm{P_A}$ can make $\mathrm{P_B}$ accept an incorrect output.

[5] This is a concrete C&C-GC-based dual-path solution to 2-output S2PC, where the circuits evaluated by each party only compute her respective output. [Kir08, §6.6] and [SS11, §1.2] conceptualized dual-path approaches in high level, but did not explain how to ensure the same input across the two executions. Other dual-path approaches have been proposed using a single GC per party (i.e., not C&C-based), but with potential leakage of one bit of information [MF06; HKE12]. A recent method [HKE13] (see comparison in §7) devised a C&C-based dual-path approach but requiring both parties to evaluate GCs with the same underlying Boolean circuit (for some 2-output functionalities this implies that GCs have the double of the size).

2.2 Bit Commitments

The BitCom approach introduced in this paper is based on several properties of (some) BitCom schemes, reviewed hereafter. A BitCom scheme [Blu83; BCC88] is a two-party protocol for committing and revealing individual bits. In a *commit* phase, it allows a *sender* to commit to a bit value, by producing and sending a BitCom value to the *receiver*. The BitCom *binds* the *sender* to the chosen bit and, initially, *hides* the bit value from the *receiver*. Then, in a *reveal* phase, the *sender* discloses a private bit-encoding (the decommitment), which allows the *receiver* to learn the committed bit and *verify* its correctness. A scheme is *XOR-homomorphic* if any pair of BitComs can be combined (under some group operation) into a new BitCom that commits the XOR of the original committed bits, and if the same can be done with the respective decommitments.

The following paragraphs describe several properties related with decommitments and trapdoors of practical BitCom schemes. For simplicity, the description focuses on a scheme based on a *square* operation with some useful collision-resistance (i.e., "claw-free" [GMR84; Dam88]) properties.

Unconditionally hiding (UH). A BitCom scheme is called UH if, before the *reveal* phase, a *receiver* with unbounded computational power cannot learn anything about the committed bit. If there is a trapdoor (known by the receiver), then it can be used to retrieve, from any BitCom, respective bit-encodings of both bits. Still, this does not reveal any information about which bit the *sender* might have committed to. A practical instantiation was used by Blum for *coin flipping* [Blu83]. There, in a multiplicative group modulo a Blum integer with factorization unknown by the *sender*, bits 0 and 1 are encoded as group-elements with Jacobi Symbol 1 or -1, respectively.[6] The *commitment* of a bit is achieved by sending the square of a random encoding of the bit. The *revealing* is achieved by sending the known square-root.

Henceforth, a XOR-homomorphic UH BitCom scheme is suggestively dubbed a *2-to-1 square scheme* if it also has the following three useful properties:

– **Proper square-roots.** *Any BitCom (dubbed* square*) has exactly two decommitments (dubbed* proper square-roots*), encoding different bits.* In the Blum integer example, each square has four square-roots, two per bit, but it is possible to define a single proper square-root per bit (e.g., the square-root whose least significant bit is equal to the encoded bit). The multiplicative group (set of residues and respective multiplication operation) can be easily adjusted to consider only proper square-roots, since the additive inverse of a non-proper square root is a proper square-root encoding the same bit.

– **From trapdoor to decommitments.** *There is a trapdoor whose knowledge allows extracting a pair of proper square-roots (the two decommitments) from any square (the BitCom).* Such pair is dubbed a *non-trivially correlated pair*, in the sense that the two proper square-roots are related but cannot

[6] A Blum integer is the product of two prime powers, where each prime is congruent with 3 modulo 4, and each power has an odd exponent. For a fixed Blum integer, the Jacobi Symbol is a completely multiplicative function that maps any group element into 1 or -1 (more detailed theory can be found, for example, in [NZM91]).

be simultaneously found (except with the help of a trapdoor). This property allows a 2-out-of-1 OT: $\mathrm{P_B}$ selects a proper square-root and sends its square to $\mathrm{P_A}$, who then uses the trapdoor to obtain the two proper square-roots. In the Blum integer example, the trapdoor is its factorization.

– **From decommitments to trapdoor.** *Any non-trivially correlated pair is a trapdoor.* This is useful for the forge-and-lose technique, as the discovery (by $\mathrm{P_B}$) of such a pair (a trapdoor of $\mathrm{P_A}$), in case $\mathrm{P_A}$ acted maliciously, is the condition that allows $\mathrm{P_B}$ to decrypt the input bits of $\mathrm{P_A}$. In the Blum integer example, its factorization can be found from any pair of proper square-roots of the same square.

Unconditionally binding (UB). A BitCom scheme is called UB if a *sender* with unbounded computational power cannot make the *receiver* accept an incorrect bit value in the *reveal* phase. If there is a trapdoor known by some party, then the party can use it to efficiently retrieve (i.e., decrypt) the committed bit from any BitCom value. A practical instantiation is the Goldwasser-Micali probabilistic encryption scheme [GM84], assuming that modulo a Blum integer it is intractable for the *receiver* to decide quadratic residuosity (of residues with Jacobi Symbol 1). A bit 1 or 0 is committed by selecting a random group element and sending its square, or sending the additive inverse of its square, respectively.[7] To decommit 1 or 0, the *sender* reveals the bit and the respective random group element, letting the *receiver* verify that its square or additive-inverse of the square, respectively, is equal to the BitCom value. The factorization of the Blum integer is a trapdoor that enables efficient decision of quadratic residuosity.

Remark. The basis of the forge-and-lose technique (§4) is a combination of UB and UH BitCom schemes, with the *sender* in the UB scheme being the *receiver* in the UH scheme, and knowing a common trapdoor for both schemes. For the Blum integer examples, and assuming intractability of deciding quadratic residuosity (without a trapdoor), this would mean using the same Blum integer in both schemes, with its factorization as trapdoor. There are known protocols to prove correctness of a Blum integer (e.g., [vdGP88]).

The two exemplified schemes are XOR-homomorphic under modular multiplication. For the purpose of the new S2PC-with-BitComs protocol (§5), this homomorphism is useful in enabling efficient ZKPs *of knowledge* (ZKPoKs) related with committed bits, and efficient negotiation of random bit-encodings and respective BitComs (emulating an ideal functionality where the *trusted third party* would select the BitComs randomly). The property is also useful for linking several S2PC executions, via ZKPs about relations between the input bits of one execution and the input and output bits of previous executions (§6).

3 The BitCom approach

This section introduces a BitCom approach that combines a BitCom setting (where there is a BitCom for each circuit input and output bit) and a C&C struc-

[7] The additive inverse of a square is necessarily a non-quadratic residue with Jacobi Symbol 1, modulo a Blum integer, because -1 has the same property.

ture (where there are several GCs, each with two keys for each input and output wire). In this approach, based on the XOR-homomorphism of UH BitComs, the consistency of input and output wire keys across different GCs is *statistically* ensured within the C&C, rather than using a ZKP of consistency.[8]

3.1 Cut-and-choose stages. The S2PC-with-BitComs protocol to be defined in this paper is built on top of a C&C approach with a Commit-Challenge-Respond-Verify-Evaluate structure. In a Commit stage, $\mathrm{P_A}$ builds and sends several GCs, as well as complementary elements (dubbed *connectors*) related with BitComs and with the circuit input and output wire keys of GCs. At this stage, $\mathrm{P_A}$ does not yet reveal the circuit input keys that allow the evaluation of each GC. Then, in the Challenge stage, $\mathrm{P_A}$ and $\mathrm{P_B}$ jointly decide a random partition of the set of GCs into two subsets, one for *verification* and the other for *evaluation*. Possibly, the subsets may be conditioned to a predefined restriction about their sizes (e.g., a fixed proportion of *verification* vs. *evaluation* GCs, or simply not letting the number of *evaluation* GCs exceed some value). In the subsequent Respond stage, $\mathrm{P_A}$ sends to $\mathrm{P_B}$ the elements that allow $\mathrm{P_B}$ to *fully verify* the correctness of the GCs selected for *verification*, to *partially verify* the connectors of all the GCs (in different ways, depending on whether they are associated with *verification* or *evaluation* challenges), and to *evaluate* the GCs (and respective connectors) selected for *evaluation*. In the Verify stage, if any verification step fails, then $\mathrm{P_B}$ aborts the protocol execution; otherwise, $\mathrm{P_B}$ establishes that there is an overwhelming probability that at least one GC (and respective connectors) selected for evaluation is correct. $\mathrm{P_B}$ finally proceeds to an Evaluate stage, evaluating the *evaluation* GCs and respective connectors, and using their results to determine the final circuit output bits and respective decommitments of output BitComs. Notice that between the Verify and Evaluate stages there is no *response* stage that could let $\mathrm{P_A}$ misbehave.

3.2 Connectors. This section develops the idea of *connectors* – structures used to sustain the integration between BitComs and the C&C structure. They are built on top of a setup where one initial UH-BitCom has been defined for each input and output wire of each party, independently of the number of GCs. Then, for each input and output wire in each GC, a connector is built to provide a (statistically verifiable) connection between the two BitCom decommitments and the respective pair of wire keys. The functionality of connectors varies with the type of wire they refer to (input of $\mathrm{P_A}$, input of $\mathrm{P_B}$, output of $\mathrm{P_B}$), as illustrated in high level in Fig. 2.

Connectors are used in a type of commitment scheme (i.e., with *commit* and *reveal* phases) that takes advantage of the C&C substrate. First, each connector is committed in the C&C Commit stage, hiding the respective two wire keys, but binding $\mathrm{P_A}$ to them and to their relation with BitCom decommitments. Then, each connector is partially revealed during the C&C Respond stage, in one of two possible complementary modes: a *reveal for verification*, related with *verification* GCs; or a *reveal for evaluation*, related with *evaluation* GCs. All

[8] The protocol still includes several efficient ZKPs related with BitComs, but they are not about the consistency of wire keys across different GCs.

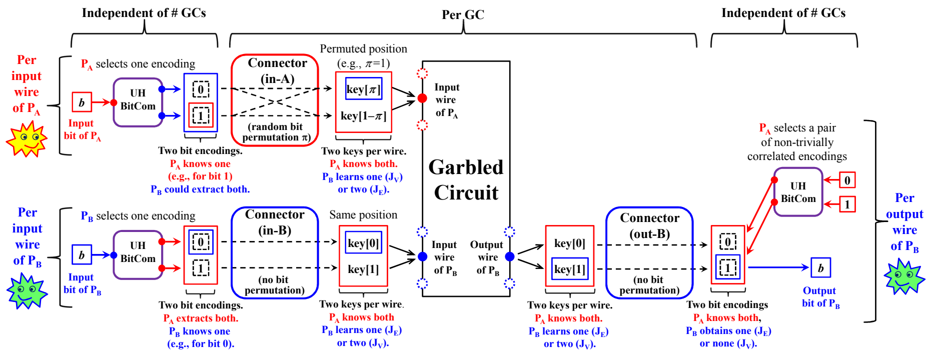
Independent of # GCs
Per GC
Independent of # GCs
Per input wire of P_A
P_A selects one encoding
b
Input bit of P_A
UH BitCom
0
1
Two bit encodings. P_A knows one (e.g., for bit 1) P_B could extract both.
Connector (in-A)
(random bit permutation π)
Permuted position (e.g., π=1)
key[π]
key[1−π]
Two keys per wire. P_A knows both. P_B learns one (J_V) or two (J_E).
Input wire of P_A
Garbled Circuit
Per input wire of P_B
P_B selects one encoding
b
Input bit of P_B
UH BitCom
Two bit encodings. P_A extracts both. P_B knows one (e.g., for bit 0).
Connector (in-B)
(no bit permutation)
Same position
key[0]
key[1]
Two keys per wire. P_A knows both P_B learns one (J_E) or two (J_V).
Input wire of P_B
Output wire of P_B
key[0]
key[1]
Two keys per wire. P_A knows both. P_B learns one (J_E) or two (J_V).
Connector (out-B)
(no bit permutation)
Two bit encodings P_A knows both, P_B obtains one (J_E) or none (J_V).
P_A selects a pair of non-trivially correlated encodings
UH BitCom
0
1
b
Output bit of P_B
Per output wire of P_B

Fig. 2: **Connectors.** Legend: $\mathrm{P_A}$ (GC *constructor*); $\mathrm{P_B}$ (GC *evaluator*); J_V and J_E (subsets of *verification* and *evaluation* GC indices, respectively); $\boxed{c}$ (group-element encoding bit c); key[c] (wire key with underlying bit c).

verifications associated with these two reveal modes are performed in the C&C VERIFY stage, when $\mathrm{P_B}$ can still, immune to selective failure attacks, complain and abort in case it finds something wrong. $\mathrm{P_A}$ never executes simultaneously the two reveal modes for the same wire of the same GC, because such action would reveal the input bits (in case of wires of $\mathrm{P_A}$) or both BitCom decommitments (i.e., the trapdoor of $\mathrm{P_A}$, in case of wires of $\mathrm{P_B}$). Nonetheless, since the commitment to the *connector* binds $\mathrm{P_A}$ to the answers that it can give in each type of reveal phase, an incorrect connector can pass undetectably at most through one type of reveal mode. Thus, within the C&C approach, there is a negligible probability that $\mathrm{P_A}$ manages to build incorrect connectors for all evaluation indices and go by undetected. The specific constructions follow:

For each input wire of $\mathrm{P_A}$:

- **Commit.** $\mathrm{P_A}$ selects a random permutation bit and a respective random encoding (a group-element dubbed *multiplier*) using the same 2-to-1 square scheme used to commit the input bits of $\mathrm{P_A}$. $\mathrm{P_A}$ uses the homomorphic group operation to obtain a new encoding (dubbed *inner encoding*) that encodes the permuted version of her input bit, and sends its square (a new inner UH BitCom) to $\mathrm{P_B}$. $\mathrm{P_A}$ then builds a commitment of each of the two wire input keys (using some other commitment scheme), one for bit 0 and the other for bit 1, and sends them to $\mathrm{P_B}$ in the form of a pair with the respective permuted order.
- **Reveal for verification.** $\mathrm{P_A}$ decommits the two wire input keys (using the *reveal* phase of the respective commitment scheme), and decommits the permutation bit (by revealing the multiplier). $\mathrm{P_B}$ uses the two wire input keys (obtained for all input wires) to verify the correctness of the GC and simultaneously obtain the underlying bit of each input key. Then, $\mathrm{P_B}$ verifies that the ordering of the bits underlying the pair of revealed input keys is consistent with the decommitted permutation bit.
- **Reveal for evaluation.** $\mathrm{P_A}$ decommits the input key that corresponds to her input bit, and decommits the permuted input bit (by revealing the inner encoding), thus allowing $\mathrm{P_B}$ to verify that it is consistent with the position of

the opened key commitment. As the value of the permuted bit is independent of the real input bit, nothing is revealed about the bit underlying the opened key. If $\mathrm{P_A}$ would instead reveal the other key, $\mathrm{P_B}$ would detect the cheating in a time when it is still safe to abort the execution and complain.

For each input wire of $\mathbf{P_B}$:

- **Commit.** $\mathrm{P_A}$ selects a pair of random encodings of bit 0 (dubbed *multipliers*) and composes them homomorphically with the two known decommitments of the original input BitCom of $\mathrm{P_B}$ (which $\mathrm{P_A}$ has extracted using the trapdoor), thus obtaining two new independent encodings (dubbed *inner encodings*, one for bit 0 and one for bit 1). $\mathrm{P_A}$ then sends to $\mathrm{P_B}$ the respective squares (dubbed *inner squares*). For simplicity, it is assumed here that the inner encodings can be directly used as input wire keys of the GC (the full version of this paper shows how to relax this assumption).
- **Reveal for verification.** $\mathrm{P_A}$ reveals the two inner encodings. $\mathrm{P_B}$ verifies that they are the proper square-roots of the received inner squares, and that they encode bits 0 and 1, respectively. Then, $\mathrm{P_A}$ uses them as the circuit input keys in the GC verification procedure, verifying their correctness. A crucial point is that the two inner encodings are proper square-roots of independent BitComs and thus do not constitute a trapdoor.
- **Reveal for evaluation.** $\mathrm{P_A}$ reveals the two multipliers. $\mathrm{P_B}$ verifies that both encode bit 0, and homomorphically verifies that they are correct (their squares lead the original BitCom into the two received inner squares). Since $\mathrm{P_B}$ knows one (and only one) decommitment of the input BitCom, it can multiply it with the respective multiplier to learn the respective inner encoding and use it as an input wire key. This procedure is resilient to selective failure attack, because both multipliers are verified for correctness, and because the two inner encodings (of which $\mathrm{P_B}$ only learns one) are statistically correct input keys (i.e., they would be detected as incorrect if they had been associated with a *verification* GC).

For each output wire of $\mathbf{P_B}$: The construction is essentially symmetric to the case of input wires of $\mathrm{P_B}$. Again for simplicity, it is assumed here that the output keys can directly be group-elements (dubbed *inner encodings*) that are proper square-roots of independent squares. The underlying bit of each output key is thus the bit encoded by it (in the role of inner encoding). $\mathrm{P_A}$ commits by initially sending the two inner squares to $\mathrm{P_B}$. Then, for *verification* challenges, from the GC verification procedure $\mathrm{P_B}$ learns 2 keys and respective underlying bits. $\mathrm{P_B}$ can verify that they are respective proper square-roots of the inner squares and that they encode the respective bits. For *evaluation* challenges, $\mathrm{P_A}$ sends only the two multipliers, and $\mathrm{P_B}$ verifies homomorphically that they are correct. Then, $\mathrm{P_B}$ learns one output key from the GC evaluation procedure, which is an inner encoding, and uses the respective multiplier to obtain the respective decommitment of the output BitCom.

The overall construction requires a number of group elements (multipliers and inner encodings) proportional to the number of input and output wires, but independent of the number of intermediate wires in the circuit.

4 The forge-and-lose technique

This section introduces a new technique, dubbed *forge-and-lose*, to improve the typical C&C-GCs-based approach, by using the BitCom approach to provide a new path for successful computation of final circuit output. More precisely, if in the EVALUATE stage there is at least one GC and respective connectors leading to a correct output (i.e., decommitments of the UH BitComs, for the correct circuit output bits), and if a malicious P_A^* successfully *forges* some other output, then P_A^* *loses* the privacy of her input bits to P_B, allowing P_B to directly use a Boolean circuit to compute the intended output. This loss of privacy is not a violation of security, but rather a disincentive against malicious behavior by P_A^*.

The forge-and-lose path significantly reduces the probabilistic gap available for malicious behavior by P_A that might lead P_B to accept an incorrect output. The technique provides up to 1 bit of statistical security per GC, which constitutes an improvement factor of about 3.1 (either in reduction of number of GCs or in increase of number of bits of statistical security) in comparison with C&C-GCs that require a majority of correct *evaluation* GCs. As noted by Lindell [Lin13], in this setting the optimal C&C partition corresponds to an independent selection of verification and evaluation challenges. Still, for some efficiency tradeoffs it may be preferable to impose some restrictions on the number of *verification* and *evaluation* challenges (e.g., ensure that there are more *verification* than *evaluation* challenges). The full version of this paper shows the error probabilities associated with different C&C partition methods.

The forge-and-lose technique is illustrated in high level in Fig. 3. It can be merged into the C&C and BitCom approach as follows:

– **Encryption scheme.** P_A encrypts her own input bits using as key the trapdoor (known by P_A) of the UH-BitCom scheme used (by P_A) to produce BitComs of the output bits of P_B. Then, P_A gives a ZKP that her encrypted input is the same as that used in the S2PC protocol, i.e., the one committed by P_A with an UH-BitCom scheme with trapdoor known by P_B. If both schemes are XOR-homomorphic (see practical example in §2.2), the ZKP can be achieved efficiently with standard techniques, namely with a statistical combination across input wires, requiring communication linear with the statistical security parameter.

– **Forge-and-lose evaluation.** In the EVALUATE stage, if a *connector* leads an output key to an invalid decommitment, then the respective GC is ignored altogether. If for the remaining GCs all connectors lead to consistent decommitments across all GCs, i.e., if for each output wire index the same valid bit-encoding (proper square-root of the output BitCom) is obtained, then P_B accepts them as correct. However, if P_A acted maliciously, there may be a forged GC and connector leading to a valid (verifiable) decommitment that is different from the decommitment obtained from another correct GC and connector, for the same output wire index. If P_B obtains any such pair of decommitments, i.e., a non-trivially correlated pair of square-roots of the same square, then P_B gets the trapdoor with which P_A encrypted her input, and follows to decrypt the input bits of P_A and use them directly to compute the correct final circuit output in the clear.

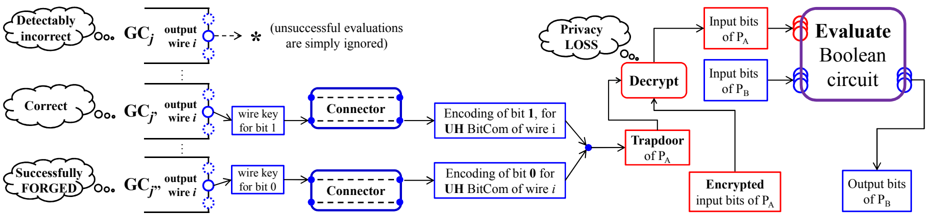

Fig. 3: **Forge-and-lose.** *Evaluation* path followed by $\mathrm{P_B}$, the evaluator of *garbled circuits* (GCs), if different GCs built by a malicious $\mathrm{P_A}$ and selected for *evaluation* (e.g., with indices j', j'') lead to valid but different decommitments of the same *unconditionally hiding* (UH) BitCom (e.g., with index i).

5 Protocol for 1-output S2PC-with-BitComs

This section describes the new C&C-GCs-based protocol for 1-output S2PC-with-BitComs, enhanced with a forge-and-lose technique. The BitComs are XOR-homomorphic, so the mentioned ZKPoKs are efficient using standard techniques.

0. Setup. The parties agree on the protocol goal, namely on a specification of a Boolean circuit whose evaluation result is to be learned privately by $\mathrm{P_B}$, on the necessary security parameters, on a C&C partitioning method, and on the necessary sub-protocols. Each party selects a *2-to-1 square* scheme, and proposes it to the other party, without revealing the trapdoor but giving a respective ZKPoK that proves the correctness of the public parameters.
1. Produce initial BitComs.
   (a) UH Commit Input Bits. Each party selects an initial UH BitCom for each of its own circuit input bits, using the 2-to-1 square scheme with trapdoor known by the other party, and sends it to the other party. $\mathrm{P_B}$ gives a ZKPoK of a valid decommitment of the respective BitComs.
   (b) UB Commit Input Bits of $\mathrm{P_A}$. $\mathrm{P_A}$ commits again to each of her input bits, now using an UB-BitCom scheme with trapdoor equal to the trapdoor (known by $\mathrm{P_A}$) of the UH-BitCom scheme used by $\mathrm{P_B}$ to commit the input bits of $\mathrm{P_B}$. $\mathrm{P_A}$ gives a ZKPoK of equivalent decommitments between the UH BitComs of the input of $\mathrm{P_A}$ (with trapdoor known by $\mathrm{P_B}$) and the UB BitComs of the input of $\mathrm{P_A}$ (with trapdoor known by $\mathrm{P_A}$), i.e., a proof that the known decommitments encode the same bits.
   (c) UH Commit Output Bits of $\mathrm{P_B}$. For each output wire index of $\mathrm{P_B}$, $\mathrm{P_A}$ selects a random encoding of bit 0 (using the UH BitCom scheme with trapdoor known by $\mathrm{P_A}$) and sends its square to $\mathrm{P_B}$. ($\mathrm{P_B}$ will find a respective decommitment only later, in the Evaluate stage.)
2. Commit. $\mathrm{P_A}$ uses her trapdoor to extract a non-trivially correlated pair of proper square-roots from each UH BitCom of the input bits (this is the so called 2-out-of-1 OT, which replaces the typical 1-out-of-2 OT used in other S2PC protocols) and output bits of $\mathrm{P_B}$. Then, $\mathrm{P_A}$ builds several GCs (in number consistent with the agreed parameters) and respective *connectors* to each input and output wire, and sends the GCs and commitments to the connectors (as specified in §3) to $\mathrm{P_B}$.

3. Challenge. The two parties use a coin-tossing sub-protocol to determine a random challenge bit for each GC, conditioned to the agreed C&C method (e.g., same number of challenges of each type, or more verification than evaluation challenges, or independent selection).[9]
4. Decide UH-BitCom Permutations. In order to emulate a *trusted third party* deciding the UH BitCom of each circuit input and output bit, both parties interact in a fully-simulatable coin-tossing sub-protocol to decide a random encoding of bit 0 for each wire index.[10] Later, each party will locally use these encodings to permute the encodings of her respective private bits, and use the square of the encodings to permute the respective UH BitComs of both parties. Given the XOR-homomorphism, the initial and the final UH BitComs commit to the same bits.
5. Respond. For each C&C challenge bit, $\mathrm{P_A}$ makes either the *reveal for verification* or the *reveal for evaluation* of the connectors, as specified in §3.
6. Verify. For *verification* indices, $\mathrm{P_B}$ obtains two keys per input wire, verifies the correctness of the GC and makes the respective partial verification of connectors (without learning the decommitments of the BitComs of output bits of $\mathrm{P_B}$). For *evaluation* indices, $\mathrm{P_B}$ makes the respective partial verification of the connectors and obtains one key per input wire. If something is found wrong, $\mathrm{P_B}$ aborts and outputs Fail.
7. Evaluate. For each *evaluation* index, $\mathrm{P_B}$ uses the one key per input wire to evaluate the GC, obtain one key per output wire and use the respective revealed part of the connector (namely, one of the two received multipliers) to obtain a decommitment (bit encoding) of the respective output BitCom. There is an overwhelming probability that there is at least one *evaluation* GC whose connectors lead to valid decommitments in all output wires. If all obtained valid decommitments are consistent across different GCs, then $\mathrm{P_B}$ accepts them as correct. Otherwise, $\mathrm{P_B}$ proceeds into the forge-and-lose path as follows. It finds a non-trivially correlated pair of square-roots and uses it as a trapdoor to decrypt the input bits of $\mathrm{P_A}$, from the respective UB BitComs. In possession of the input bits of both parties, $\mathrm{P_B}$ directly evaluates the final circuit output. Then, from within the decommitments already obtained from the *evaluation* connectors, $\mathrm{P_B}$ finds the output bit encodings that are consistent with the circuit output bits, and accepts them as the correct ones. This marks the end of the forge-and-lose path.
8. Apply BitCom Permutations. Each party applies the previously decided random permutations to the encodings of the respective circuit input and output bits, and applies the square of the random encodings as permutations to the UH BitComs of the circuit input and output bits of both parties.

[9] The standalone coin-tossing does not need to be fully simulatable, but the proof of security takes advantage of the ability of the simulated $\mathrm{P_A}$ (with rewinding access to a possibly malicious $\mathrm{P_B^*}$) to decide the outcome of the coin-toss. Subtle alternatives would be possible, depending on some changes related with the remaining stages.

[10] To achieve simulability of the overall protocol under each possible malicious party ($\mathrm{P_A^*}$ and $\mathrm{P_B^*}$), the simulator of this coin-tossing needs to be able to induce the final BitComs in the real world to be equal to those decided by the trusted third party in the ideal world, and at the same time deal with a probabilistic possibility of abort dependent on those final BitCom values (e.g., see [Lin03]).

9. FINAL OUTPUT. Each party privately outputs her circuit input and output bits and the respective final encodings, and also outputs the (commonly known) final UH BitComs of the circuit input and output bits of both parties. $\mathrm{P_A}$ outputs even if $\mathrm{P_B}$ aborts at any time after the APPLY BITCOM PERMUTATIONS stage.

Remark. When using the 1-output protocol within larger protocols, care needs to be taken so that $\mathrm{P_A}$ cannot distinguish between $\mathrm{P_B}$ having learned his output via the normal evaluation path vs. via the forge-and-lose path.

6 Discussion

6.1 Complexity. Besides the computation and communication related with (the reduced number of) GCs, the new S2PC-with-BitComs protocol requires instantiating the connectors (which brings a cost proportional to the number of input and output wires, multiplied by the number of GCs), performing ZKPoKs related with BitComs and to prove correctness of the BitCom scheme parameters, and performing secure two-party coin-tossing (which is significant for the decision of random BitComs values). Based on the XOR-homomorphism, the ZKPoKs related with input wires can be parallelized efficiently with standard techniques, with a communication cost linear in a statistical parameter but independent of the number of input wires, though with computational cost proportional to the product of the statistical parameter and the number of input wires.

With an instantiation based on Blum integers, the inversion of an UH BitCom using the trapdoor (i.e., computing a modular square-root) is approximately computationally equivalent to one exponentiation modulo each prime factor. Thus, besides proving correctness of the Blum integer (which can be achieved with a number of exponentiations that is linear in the statistical parameter), and performing a fully-simulatable coin-tossing sub-protocol to decide random BitCom permutations (which can be instantiated with a number of exponentiations that is linear in the number of input and output wires, and performed in a group of smaller order), the 1-output S2PC-with-BitComs protocol only requires a number of exponentiations that is linear in the number of input wires of $\mathrm{P_B}$, and only computed by $\mathrm{P_A}$. This is in contrast with other protocols whose required number of exponentiations by both parties is proportional to the number of GCs multiplied by the number of input wires (e.g., [LP11]), though in compensation those exponentiations are supported in groups with smaller moduli length and sub-groups of smaller order.

The protocol can be optimized in several ways. For example, with a *random seed checking* (RSC) technique [GMS08] the communication of elements (including GCs and connectors) associated with verification challenges can be replaced by the sending and verification of small random seeds (used to pseudo-randomly generate the elements) and a commitment (to the elements). The technique can be applied independently to GCs and connectors, and can also be used to reduce some of the communication corresponding to connectors associated with *evaluation* challenges. As another example, some group elements used in connectors of $\mathrm{P_A}$ can be reduced in size, since their binding properties only need to hold during the execution of the protocol.

Concrete results. An analytic estimation of communication complexity is made in the full version of this paper (ignoring overheads due to communication protocols), for two different circuits: an AES-128 circuit with 6,800 multiplicative gates [Bri13] and 128 wires for the input of each party and for the output of $\mathrm{P_B}$; and a SHA-256 circuit with 90,825 multiplicative gates [Bri13] and 256 wires for the input of each party and output of $\mathrm{P_B}$.

An interesting metric is the proportional overhead of communicated elements beyond GCs (i.e., connectors, BitComs and associated proofs) in comparison with the size occupied only by the GCs. For 128 bits of cryptographic security, instantiated with 3,072-bit Blum integers [BBB+12], and 40 bits of statistical security achieved using 41 GCs of which at most 20 are for evaluation, the estimated overhead is about 55% and 8%, for the AES-128 and SHA-256 circuits, respectively, without the RSC technique applied to the GCs. This metric gives an intuition about the communication cost inherent to the BitCom approach, but is not good enough on its own. For example, when applying the RSC technique also at the level of GCs, the overall communication is reduced significantly, but (because the size corresponding to GCs is reduced) the proportional overhead increases to 158% and 23%, respectively. Nonetheless, even these overheads are low when compared to the cost associated with the additional GCs needed in a C&C that requires a majority of correct evaluation GCs (i.e., on its own an overhead of about 200%, and asymptotically up to about 210%). Clearly, the proportional overhead decreases with the ratio given by the number of input and output wires divided the number of multiplicative gates.

There are other optimizations and C&C configurations that reduce the communication even more, with tradeoffs with computational complexity. For example, by restricting the number of *evaluation* GCs to be at most 8, but increasing the overall number of GCs to 123 (this was the minimal number of GCs required by the typical C&C to achieve 40 bits of statistical security), the estimated communication complexity is approximately of the order of 62 million bits and 418 million bits, respectively for the exemplified circuits. A *pipelining* technique [HEKM11] could also be considered, such that the garbled-gates are not all stored in memory at the same time. This would increase the computation by $\mathrm{P_A}$, but not affect the amount of communicated elements.

6.2 Linked executions. A simple example of linked executions is the mentioned dual-path execution approach, where each party reuses the same input bits (and BitComs) in two different executions. Furthermore, it may be useful to achieve more general linkage, such as proving that the private input bits of a S2PC satisfy certain *non-deterministic polynomial* verifiable relations with the private input and output bits of previous S2PCs. Based on the XOR-homomorphism of BitComs, this can be proven with efficient ZKPs. For example, proving that a certain BitCom commits to the NAND of the bits committed by two other BitComs can be reduced to a simple ZKP that there are at least two 1's committed in a triplet of BitComs, with the triplet being built from a XOR-homomorphic combination of the original three BitComs.[11]

For example, since Boolean circuits can be implemented with NAND gates alone, it is possible to prove, outside of the GCs, those transformations and relations that involve only the bits of one party. For example, for protocols defined as a recursion of small GC-based S2PC sub-protocols in the semi-honest model (e.g., [LP02]), security can be enhanced to resist also the malicious model, by simply (1) replacing each GC with a C&C-GCs with BitComs, and (2) by naturally using the input and output of previous executions (or transformations thereof) as the input of the subsequent executions.

6.3 Security. The protocol can be proven secure in the plain model (i.e., without hybrid access to ideal functionalities), assuming the simulator has black-box access with rewindable capability to a real adversary. The simulator is able to extract the input of the malicious party in the real world from the respective ZKPoKs of decommitments, and thus hand it over to the *trusted third party* in the ideal world. The two-party coin tossing used to select random permutations of group-elements needs to be fully-simulatable, because the final BitComs and decommitments are also part of the final output of honest parties. Subtle changes are needed to the ideal functionality when the protocol is adjusted to the 2-output case where each party learns a private circuit output. Achieving security in the *universal composability* model [CLOS02] is left for future work.

7 Related work

7.1 Two other optimal C&C-GCs

Two recently proposed C&C-GCs-based protocols [Lin13; HKE13] also minimize the number of GCs, requiring only that at least one evaluation GC is correct.

Lindell [Lin13] enhances a typical C&C-GCs-based protocol by introducing a second C&C-GCs, dubbed *secure-evaluation-of-cheating* (SEOC), where $\mathrm{P_B}$ recovers the input of $\mathrm{P_A}$ in case $\mathrm{P_B}$ can provide two different garbled output values from the first C&C-GCs. The concept of input-recovery resembles the forge-and-lose technique, but the methods are quite different. For example, the SEOC phase requires interaction between the parties after the first GC evaluation phase, whereas in the forge-and-lose the input-recovery occurs offline.

Huang, Katz and Evans [HKE13] propose a method that combines the C&C-GCs approach with a verifiable secret sharing scheme (VSSS). The parties play different roles in two symmetric C&C-GCs, and then securely compare their outputs. This requires the double of GCs, but in parallel across the two parties. By requiring a predetermined number of verification challenges, the necessary number of GCs is only logarithmically higher than the optimal that is achieved with an independent selection of challenges. In their method, the deterrent against

[11] The first bit is the NAND of the two last if and only if there are at least two 1's in the triplet composed of the first bit and of the XOR of the first bit with each of the other two bits [Bra06]. A different method can be found in [BDP00].

optimal malicious GCs construction does not involve the GC constructor party having her input revealed to the GC evaluator.

In the SEOC and VSSS descriptions, the method of ensuring input consistency across different GCs is supported on discrete-log based intractability assumptions. The descriptions do not consider general linkage of S2PC executions related with output bits, but their input bits are also committed using XOR-homomorphic BitComs. In contrast, the S2PC-with-BitComs described in this paper, with an instantiation based on Blum integers, is based on intractability of deciding quadratic residuosity and requires a lower number of exponentiations, though with each exponentiation being more expensive due to the larger size of group elements and group order, for the same cryptographic security parameter. Future work may better clarify the tradeoffs between the three techniques.

7.2 Other related work

Jarecki and Shmatikov [JS07] described a S2PC protocol with committed inputs, using a single verifiably-correct GC, but with the required number of exponentiations being linear in the number of gates. In comparison, the protocol in this paper allows garbling schemes to be based on symmetric primitives (e.g., block-ciphers, whose greater efficiency over-compensates the cost of multiple GCs in the C&C), and the required number of exponentiations to be linear in the number of circuit input and output bits and in the statistical parameter.

Nielsen and Orlandi proposed LEGO [NO09], and more recently Frederiksen et al. proposed Mini-Lego [FJN$^+$13], a fault-tolerant circuit design that computes correctly even if some garbled gates are incorrect. Their protocol, which uses a cut-and-choose at the garbled-gate level (instead of at the GC level) to ensure that most garbled gates used for evaluation are correct, requires a single GC but of larger dimension. It would be interesting to explore, in future work, how to integrate a forge-and-lose technique into their cut-and-chose at the gate level.

Kolesnikov and Kumaresan [KK12] described a S2PC slice-evaluation protocol, based on information theoretic GCs, allowing the input of one GC to directly use the output of a previous GC. Their improvements are valid if the linked GCs are shallow, and if one party is semi-honest and the other is covert. In contrast, the S2PC-with-BitComs protocol in this paper allows any circuit depth and any party being malicious.

Nielsen et al. [NNOB12] proposed an OT-based approach for S2PC, potentially more efficient than a C&C-GCs if network latency is not an issue. However, the number of communication rounds of their protocol is linear in the depth of the circuit, thus being outside of the scope of this paper (restricted to C&C-GCs-based protocols with a constant number of communication rounds).

Acknowledgments. The author thanks: his Ph.D. co-advisor Alysson Bessani, for the valuable discussions and suggestions that contributed to improve the presentation of an earlier version of this paper; another reviewer, who wished to remain anonymous, for valuable suggestions that also contributed to improve the presentation of an earlier version of this paper; the anonymous reviewers of CRYPTO 2013 and ASIACRYPT 2013 conferences, for their useful comments.

References

[BBB+12] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management – Part 1: General (Revision 3) – NIST Special Publication 800-57. U.S. Department of Commerce, NIST-ITL-CSD, July 2012. 16

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988. 7

[BDP00] J. Boyar, I. Damgård, and R. Peralta. Short Non-Interactive Cryptographic Proofs. *J. Cryptology*, 13:449–472, 2000. 17

[Bea96] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proc. STOC '96*, pages 479–488. ACM, New York, 1996. 5

[BHR12] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *Proc. CCS '12*, pages 784–796. ACM, New York, 2012. See also Cryptology ePrint Archive, Report 2012/265. 4, 5

[Blu83] M. Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15:23–27, January 1983. 7

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proc. STOC '90*, pages 503–513. ACM, New York, 1990. 4

[Bra06] L. T. A. N. Brandão. A Framework for Interactive Argument Systems using Quasigroupic Homorphic Commitment. Cryptology ePrint Archive, Report 2006/472, 2006. 17

[Bra13] L. T. A. N. Brandão. Secure Two-Party Computation with Reusable Bit-Commitments, via a Cut-and-Choose with Forge-and-Lose Technique (Technical Report). Cryptology ePrint Archive, 2013. 1, 4

[Bri13] Bristol Cryptography Group. Circuits of Basic Functions Suitable For MPC and FHE. `http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/`, Accessed June 2013. 16

[Can00] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *J. Cryptology*, 13:143–202, 2000. See also Cryptology ePrint Archive, Report 1998/018. 2

[CGT95] C. Crépeau, J. v. d. Graaf, and A. Tapp. Committed Oblivious Transfer and Private Multi-Party Computation. In D. Coppersmith, editor, *CRYPTO '95*, vol. 963 of *LNCS*, pages 110–123. Springer-Verlag, 1995. 5

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proc. STOC '02*, pages 494–503. ACM, New York, 2002. See also Cryptology ePrint Archive, Report 2002/140. 17

[Dam88] I. B. Damgård. *The application of claw free functions in cryptography*. PhD thesis, Aarhus University, Mathematical Institute, 1988. 7

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28:637–647, June 1985. 5

[FJN+13] T. Frederiksen, T. Jakobsen, J. Nielsen, P. Nordholt, and C. Orlandi. MiniLEGO: Efficient Secure Two-Party Computation from General Assumptions. In T. Johansson and P. Nguyen, editors, *EUROCRYPT '13*, vol. 7881 of *LNCS*, pages 537–556. Springer-Verlag, 2013. See also Cryptology ePrint Archive, Report 2013/155. 18

[FN13] T. Frederiksen and J. B. Nielsen. Fast and Maliciously Secure Two-Party Computation Using the GPU. In M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini, editors, *ACNS '13*, vol. 7954 of *LNCS*, pages 339–356. Springer-Verlag, 2013. 2

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. 8

[GMR84] S. Goldwasser, S. Micali, and R. L. Rivest. A "Paradoxical" Solution To The Signature Problem. In *Proc. FOCS '84*, pages 441–448. IEEE Computer Society, 1984. 7

[GMS08] V. Goyal, P. Mohassel, and A. Smith. Efficient Two Party and Multi Party Computation Against Covert Adversaries. In N. Smart, editor, *EUROCRYPT '08*, vol. 4965 of *LNCS*, pages 289–306. Springer-Verlag, 2008. 4, 15

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *Proc. STOC '87*, pages 218–229. ACM, New York, 1987. 4

[Gol04] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, chapter 7 edition, 2004. 1

[HEKM11] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *Proc. SEC '11*. USENIX Association, 2011. 4, 16

[HKE12] Y. Huang, J. Katz, and D. Evans. Quid-Pro-Quo-tocols: Strengthening Semi-Honest Protocols with Dual Execution. In *Proc. S&P '12*, may 2012. 6

[HKE13] Y. Huang, J. Katz, and D. Evans. Efficient Secure Two-Party Computation Using Symmetric Cut-and-Choose. In R. Canetti and J. Garay, editors, *CRYPTO '13*, vol. 8043 of *LNCS*, pages 18–35. Springer-Verlag, 2013. See also Cryptology ePrint Archive, Report 2013/081. 4, 6, 17

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending Oblivious Transfers Efficiently. In D. Boneh, editor, *CRYPTO '03*, vol. 2729 of *LNCS*, pages 145–161. Springer-Verlag, 2003. 5

[JS07] S. Jarecki and V. Shmatikov. Efficient Two-Party Secure Computation on Committed Inputs. In M. Naor, editor, *EUROCRYPT '07*, vol. 4515 of *LNCS*, pages 97–114. Springer-Verlag, 2007. 18

[Kir08] M. S. Kiraz. *Secure and Fair Two-Party Computation*. Phd thesis, Technische Universiteit Eindhoven, Netherlands, 2008 2008. 6

[KK12] V. Kolesnikov and R. Kumaresan. Improved Secure Two-Party Computation via Information-Theoretic Garbled Circuits. In I. Visconti and R. De Prisco, editors, *SCN '12*, vol. 7485 of *LNCS*, pages 205–221. Springer-Verlag, 2012. 5, 18

[Kol09] V. Kolesnikov. Advances and impact of secure function evaluation. *Bell Labs Technical Journal*, 14(3):187–192, 2009. 2

[KS06] M. S. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In *Proc. 27th Symp. Information Theory in the Benelux*, pages 283–290, 2006. 5, 6

[KS08a] M. S. Kiraz and B. Schoenmakers. An Efficient Protocol for Fair Secure Two-Party Computation. In T. Malkin, editor, *CT-RSA '08*, vol. 4964 of *LNCS*, pages 88–105. Springer-Verlag, 2008. 6

[KS08b] V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In L. Aceto, I. Damgård, L. Goldberg, M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP '08*, vol. 5126 of *LNCS*, pages 486–498. Springer-Verlag, 2008. 4

[KSS12] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *Proc. Security '12*, pages 285–300. USENIX Association, 2012. See also Cryptology ePrint Archive, Report 2012/179. 2

[Lin03] Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. *J. Cryptology*, 16(3):143–184, 2003. See also Cryptology ePrint Archive, Report 2001/107. 14

[Lin13] Y. Lindell. Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries. In R. Canetti and J. Garay, editors, *CRYPTO '13*, vol. 8043 of *LNCS*, pages 1–17. Springer-Verlag, 2013. See also Cryptology ePrint Archive, Report 2013/079. 4, 12, 17

[LP02] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. *J. Cryptology*, 15(3):177–206, 2002. 2, 17

[LP07] Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In M. Naor, editor, *EUROCRYPT '07*, vol. 4515 of *LNCS*, pages 52–78. Springer-Verlag, 2007. See also Cryptology ePrint Archive, Report 2008/049. 6

[LP09] Y. Lindell and B. Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *J. Cryptology*, 22(2):161–188, 2009. 5

[LP11] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Y. Ishai, editor, *TCC '11*, vol. 6597 of *LNCS*, pages 329–346. Springer-Verlag, 2011. See also Cryptology ePrint Archive, Report 2010/284. 5, 6, 15

[MF06] P. Mohassel and M. Franklin. Efficiency Tradeoffs for Malicious Two-Party Computation. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC '06*, vol. 3958 of *LNCS*, pages 458–473. Springer-Verlag, 2006. 6

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A New Approach to Practical Active-Secure Two-Party Computation. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO '12*, vol. 7417 of *LNCS*, pages 681–700. Springer-Verlag, 2012. See also Cryptology ePrint Archive, Report 2011/091. 5, 18

[NO09] J. B. Nielsen and C. Orlandi. LEGO for Two-Party Secure Computation. In O. Reingold, editor, *TCC '09*, vol. 5444 of *LNCS*, pages 368–386. Springer-Verlag, 2009. See also Cryptology ePrint Archive, Report 2008/427. 6, 18

[NP01] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA '01*, pages 448–457. SIAM, Philadelphia, 2001. 5

[NPS99] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proc. EC '99*, pages 129–139. ACM, New York, 1999. 4

[NZM91] I. M. Niven, H. S. Zuckerman, and H. L. Montgomery. *An introduction to the theory of numbers*. Wiley, fifth edition, 1991. 7

[Pin03] B. Pinkas. Fair Secure Two-Party Computation. In E. Biham, editor, *EUROCRYPT '03*, vol. 2656 of *LNCS*, pages 647–647. Springer-Verlag, 2003. 5

[PSSW09] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure Two-Party Computation Is Practical. In M. Matsui, editor, *ASIACRYPT '09*, vol. 5912 of *LNCS*, pages 250–267. Springer-Verlag, 2009. See also Cryptology ePrint Archive, Report 2009/314. 2, 4, 6

[Rab81] M. O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Harvard University, Aiken Computation Lab, Cambridge, MA, 1981. See typesetted version in Cryptology ePrint Archive, Report 2005/187. 5

[SS11] A. Shelat and C.-h. Shen. Two-Output Secure Computation with Malicious Adversaries. In K. Paterson, editor, *EUROCRYPT '11*, vol. 6632 of *LNCS*, pages 386–405. Springer-Verlag, 2011. See also Cryptology ePrint Archive, Report 2011/533. 3, 6

[vdGP88] J. van de Graaf and R. Peralta. A Simple and Secure Way to Show the Validity of Your Public Key. In C. Pomerance, editor, *CRYPTO '87*, vol. 293 of *LNCS*, pages 128–134. Springer-Verlag, 1988. 8

[Woo07] D. P. Woodruff. Revisiting the Efficiency of Malicious Two-Party Computation. In M. Naor, editor, *EUROCRYPT '07*, vol. 4515 of *LNCS*, pages 79–96. Springer-Verlag, 2007. See also Cryptology ePrint Archive, Report 2006/397. 6

[Yao82] A. C. Yao. Protocols for secure computations. In *Proc. FOCS '82*, pages 160–164. IEEE Computer Society, 1982. 1

[Yao86] A. C.-C. Yao. How to generate and exchange secrets. *FOCS '86*, 0:162–167, 1986. 4