Hashing Garbled Circuits for Free

Xiong Fan[1], Chaya Ganesh[2], and Vladimir Kolesnikov[3]

[1] Cornell University, Ithaca, NY, USA. xfan@cs.cornell.edu
[2] New York University, New York, NY, USA. ganesh@cs.nyu.edu
[3] Bell Labs, Murray Hill, NJ, USA. kolesnikov@research.bell-labs.com

Abstract. We introduce *Free Hash*, a new approach to generating Garbled Circuit (GC) hash at no extra cost during GC generation. This is in contrast with state-of-the-art approaches, which hash GCs at computational cost of up to 6× of GC generation. GC hashing is at the core of the cut-and-choose technique of GC-based secure function evaluation (SFE).

Our main idea is to intertwine hash generation/verification with GC generation and evaluation. While we *allow* an adversary to generate a GC $\widehat{\mathsf{GC}}$ whose hash collides with an honestly generated GC, such a $\widehat{\mathsf{GC}}$ w.h.p. will fail evaluation and cheating will be discovered. Our GC hash is simply a (slightly modified) XOR of all the gate table rows of GC. It is compatible with Free XOR and half-gates garbling, and can be made to work with many cut-and-choose SFE protocols.

With today's network speeds being not far behind hardware-assisted fixed-key garbling throughput, eliminating the GC hashing cost will significantly improve SFE performance. Our estimates show substantial cost reduction in typical settings, and up to factor 6 in specialized applications relying on GC hashes.

We implemented GC hashing algorithm and report on its performance.

Remark. In this work, we assume existence of a hash function that satisfies both correlation robustness and collision resistance. A recent work by Guo et al. [GKWY19] pointed out that our instantiation of the hash function is not collision resistant, and further they showed that it is significantly more expensive than previously thought to construct such hash functions. This significantly impacts performance of Free Hash. Additionally, a more streamlined implementation of standard hashing further reduces the performance improvements of Free Hash [Wan18]. As a result, we don't believe Free Hash improves on the standard GC-based 2PC techniques, with the current state of affairs.

1 Introduction

Today Garbled Circuit (GC) is the main technique for secure computation. It has advantages of high performance, low round complexity/low latency, and, importantly, relative engineering simplicity. Both core GC (garbling), as well as the meta-protocols, such as Cut-and-Choose (C&C), have been thoroughly investigated and are today highly optimized. Particularly in the semi-honest model there have been few asymptotic/qualitative improvements since the original protocols of Yao [Yao86] and Goldreich et al. [GMW87]. Possibly the most important development in the area of practical SFE since the 1980s was the very efficient oblivious transfer (OT) extension technique of Ishai et al. [IKNP03]. This allowed the running of an arbitrarily large number of OTs by executing a small (security parameter) number of (possibly inefficient) "bootstrapping" OT instances and a number of symmetric key primitives. The cheap OTs made a dramatic difference for securely computing functions with large inputs relative to the size of the function, as well as for GMW-like approaches, where OTs are performed in each level of the circuit. Another important GC core improvement is the Free-XOR algorithm [KS08a], which allowed for the evaluation of all XOR gates of a circuit without any computational or communication costs.

As SFE moves from theory to practice, even "small" factor improvements can have a significant effect.

1.1 Motivation of efficient GC hashing: cut-and-choose (C&C) and other uses.

In this work we improve (actually show how to achieve it for free) a core garbling feature of GC, circuit hashing. We discuss how this improves standard GC-based SFE protocols. We also discuss evaluation of certified functions, and motivate this use case.

GC hashing is an essential tool for C&C, and is employed in many uses of C&C. We start with describing C&C at the high level.

C&C. According to the "Cut-and-Choose Protocol" entry of the Encyclopedia of Cryptography and Security [TJ11], a (non-zero-knowledge) C&C protocol was first mentioned in the protocol of Rabin [Rab77] where this concept was used to convince a party that the other party sent it a specially formed integer n. The expression "cut and choose" was introduced later by Chaum in [BCC88] in analogy to a popular cake-sharing problem: given a cake to be divided among two distrustful players, one of them cuts the cake in two shares, and lets the other one choose.

Recall, the basic GC protocol is not secure against cheating GC generator, who can submit a maliciously garbled circuit. Today, C&C is the standard tool in achieving malicious security in secure computation. At the high level, it proceeds as follows. GC generator generates a number of garbled circuits $\mathsf{GC}_1, ..., \mathsf{GC}_n$ and sends them to GC evaluator, who chooses a subset of them (say, half) at random to be opened (with the help of the generator) and verifies the correctness of circuit construction. If all circuits were constructed correctly, the players proceed to securely evaluate the unopened circuits, and take the majority output. It is easy to see that the probability of GC generator succeeding in submitting a maliciously garbled circuit is exponentially small in n. We note that significant improvement in the concrete values of n required for a specific probability guarantee was achieved by relatively recent C&C techniques [LP11,Lin13,HKE13,Bra13,LR14,HKK$^+$14,AO12,KM15].

Using GC hashing for C&C. What motivates our work is the following natural idea, which was first formalized in Goyal et al. [GMS08]. To save on communication (usually a more scarce resource than computation), GC generator, firstly, generates all the circuits $\mathsf{GC}_1, ..., \mathsf{GC}_n$ from PRG seeds $s_1, ..., s_n$. Then, instead of sending the circuits $\mathsf{GC}_1, ..., \mathsf{GC}_n$, it sends their hashes $H(\mathsf{GC}_1), ..., H(\mathsf{GC}_n)$. Finally, while the evaluation circuits will need to be sent in full over the network, only the seeds $s_1, ..., s_n$ need to be sent to verify that GC generator did not cheat in the generation of the opened circuits, saving a significant amount of communication at the cost of computing and checking $H(\mathsf{GC}_i)$ for all n circuits.

On many of today's computing architectures (e.g. Intel PC CPUs, with or without hardware AES), the cost of hashing the GC can be up to $6\times$ greater than the cost of fixed-key garbling. At the same time, today's network speeds are comparable in throughput with hardware-assisted fixed-key garbling (see our calculations in Section 5.3). Hence, eliminating the GC hashing cost will improve SFE performance by eliminating the (smaller of the) cost of hashing or sending the open circuits. We stress that the use of our Free Hash requires syntactic changes in C&C protocols and it provides a security guarantee somewhat distinct from collision-resistant hash. Hence its use in C&C protocols should be evaluated for security. See Section 5.1 for more details.

Additionally, we show that a new computation/communication cost ratio offered by our free GC hash will allow for reduced communication, computation, and execution time, while achieving the same cheating probability.

SFE of private certified functions. One advantage offered by GC is the hiding of the evaluated function from the evaluator. To be more precise, the circuit topology of the function is revealed, but this information leakage can be removed or mitigated by using techniques such as universal circuit [Val76,KS08b,LMS16,KS16] or circuit branch overlay [KKW16].

In practical scenarios, evaluated functions are to be selected as allowed by a mutually agreed policy, e.g., to prevent evaluation of identity function outputting player's private input. Then evaluating a hidden function presumes either a semi-honest GC generator, or employing a method for preventing/deterring out-of-policy GC generation. An efficient C&C approach does not seem to help prevent cheating here, since check circuits will reveal the evaluated function and will not be acceptable to the GC generator. Further, depending on policy/application, the zero-knowledge proofs of correctly constructing the circuits may be very expensive.

In many scenarios, Certificate Authorities (CA) may be used to certify the correct generation of GCs. Indeed, this is quite feasible at small to medium scale. Our motivating application here is the private attribute-based credential (ABC) checking. Very recent concurrent works [CGM16,KKL$^+$16] showed for the first time that ABCs can be based on GCs. While both [CGM16,KKL$^+$16] discuss public policy only, their GC-based constructions will not preclude achieving private policy. We note that this is a novel property in the ABC literature, where all previous work (in addition to supporting very small policies only) relied in an essential manner on the policy being known to both prover and verifier.

At the high level, the architecture/steps for evaluation of private CA-certified functions is as follows.

1. CA generates seeds $s_1, ...s_n$ and, for $i = 1, ...n$, CA generates GCs GC_i, GC hashes $H(\mathsf{GC}_i)$ and signatures $\sigma_i = \mathsf{Sign}_{CA}(H(\mathsf{GC}_i))$. It sends all $s_i, H(\mathsf{GC}_i), \sigma_i$ to ABC verifier V.
2. Prover P and V proceed with execution of the ABC protocols [CGM16,KKL+16], with the following modification:
   (a) Whenever GC GC_i needs to be sent by V, instead V generates GC_i from s_i and sends to P the pair $(\mathsf{GC}_i, \sigma_i)$.
   (b) P computes $H(\mathsf{GC})$ and verifies the signature σ_i prior to continuing. If the verification or GC evaluation fails, P outputs `abort`.

Free Hash will allow to significantly (up to factor 6) reduce the computational effort required by the CA to support such an application. Indeed the cost of the signature generation can be small and ignored in cases where the signed circuits are large, or a single signature can certify a number of circuits. The latter would be the case where two players may be expected to evaluate a number of circuits.

Importantly, evaluation of certified functions may be essential in scenarios where legislative and/or operational demands require high degree of accountability and auditability (recall, digital signatures are a recognized legal instrument in many countries [Wik]). These scenarios may frequently arise in government, intelligence or military applications.

Technical results of this work will have direct impact, up to factor 6 improvement, in the bottleneck (CA load) in many scenarios discussed above.

1.2 Our Contributions and Outline of the Work

We start the presentation with a brief discussion of related work and then providing a high-level technical overview of our approach. Then, in Section 2, we introduce existing definitions and constructions required for this work. In Section 3 we discuss definitional aspects, assumptions and parameter choices of our work.

We start technical Section 4 with introducing our proposed definition of GC hash security. Our definition is weaker than the standard hash collision guarantees, yet it is possible to make free hashing work with several standard GC constructions (cf. Section 5.1 for discussion about its C&C use). We then present hashed garbling algorithms for standard garbling (based on Just Garble of [BHKR13]) as well as for half-gates garbling of [ZRE15]. Our main contribution is the improvement of the state-of-the-art half-gates; we consider hashed Just Garble a valuable generalization and an instructional example.

In Section 5, we discuss the impact of Free Hash garbling and C&C. We report on our implementation and its performance evaluation. We discuss the application to certified circuits. We propose a unified cost metric (time) and show higher speeds/smaller computation and communication for the same error probability. We estimate total execution time reduction of about 43% for the C&C components of [LP11], and of about 64% for [AO12,KM15] in settings we consider (1Gbps channel and hardware AES).

1.3 Technical Overview of Free GC Hash

In this section we present the main intuition behind our technical approach.

We take advantage of the observation that the input to the hash is a garbled circuit GC, which must be evaluatable using the garbled circuit Eval function. We will not require standard hash collision resilience of GC strings, achieving which is very costly relative to the cost of GC generation. Instead, we guarantee that if an adversary can find another string $\widehat{\mathsf{GC}}$ that matches the hash of a correctly garbled GC, then with high probability, the garbled circuit property of $\widehat{\mathsf{GC}}$ is broken and its evaluation will fail.

We present our intuition iteratively; we start with a naive efficient approach, which we then refine and arrive at a secure hashed garbling. Recall, we start with a correctly generated GC GC with the set of output decoding labels d. Adversary's goal is to generate a circuit $\widehat{\mathsf{GC}}$ with the same hash as GC, and which will not fail evaluation/decoding given *the same* output labels d. This hash guarantee is sufficient for certain GC-based SFE protocols. A syntactic difference with [GMS08] C&C hashing is that verification of Free Hash involves GC evaluation, and is only possible once input labels are received (e.g., after OT of input labels). More importantly, Free Hash, as applied to C&C, provides a security guarantee subtly distinct from collision-resistant hash. Hence, drop-in replacement of [GMS08] C&C hashing with Free Hash may not be always possible, and in general should be done by hand and original proofs re-checked. See Section 5.1 for additional discussion.

We present the intuition for the classical four-row GC; we use similar ideas to achieve half-gates GC hashing as well. We present and prove secure both Free Hash constructions.

The first Free Hash idea is to simply set the hash of the garbled circuit to be the XOR of all garbled table (GT) rows of GC. This is clearly problematic, since a cheating garbler $\mathcal{A}$ can mount, for example, the following attack. $\mathcal{A}$ will set one GT entry to be the encryption of the wrong wire label. This affects the XOR hash as follows $H(\widehat{\mathsf{GC}}) = H(\mathsf{GC}) \oplus \Delta$. Now suppose the garbler knows (or guesses) which GT entry anywhere in GC will not be used in evaluation (inactive GT row). Now $\mathcal{A}$ simply replaces the inactive GT row X with value $X \oplus \Delta$. This will restore the hash to the desired value, and since this entry will not be used in the evaluation, the garbler will not be caught.

The following refinement of this approach counters the above attack: we make the gate's output wire key depend (in an efficient manner) on *all* GT rows of that gate. The idea is that XOR hash correction, such as above, will necessarily involve modification to an active GT row, which will affect the computed wire key on that gate. Importantly, because wire keys and GT rows are related via a random (albeit known) function, a GT row offset by Δ (needed to "fix" the hash) will result in effectively randomizing the output wire label of the gate. Because a non-failing evaluation requires output wire labels to be consistent with the fixed decoding information d, $\mathcal{A}$ will now be stuck.

We attempt this by starting with a secure garbling scheme $\mathcal{G}$, and modifying the way the wire labels are defined, as follows. The two wire labels w_i^0, w_i^1

associated with gate G_i's output wire will now be treated as temporary labels. A label W_i^j of the new scheme will be obtained from the w_i^j simply by XORing it with all the GT rows of G_i.

This is not quite sufficient, as it still allows the attacker to modify a GT row and then correct it within *the same* gate table. This is possible since a "fix" for the hash does not disrupt the validity of the wire label, as both the hash and the new wire label are defined in the same manner (as XOR of all the GT rows of G_i). Our final idea, is to use the GT rows as XOR pads in a different manner for computing the GC hash and for offsetting the wire values. This way, the fix for the hash w.h.p. will not simultaneously keep the wire label valid. We achieve this by malleating GT rows prior to using them as XOR pads in wire value computation.

It is not hard to show that the above changes preserve the privacy and authenticity properties of the garbling scheme.

We summarize the intuition for the hash security of the above construction. Consider a $\widehat{\mathsf{GC}} \neq \mathsf{GC}$ that collides under the above hash. Then, the evaluation of $\widehat{\mathsf{GC}}$ will deviate from that of GC w.r.t. some wire label. Importantly, $\widehat{\mathsf{GC}}$ evaluation can subsequently either return to a valid wire label or to a correct running hash, but not both. Thus, evaluation of $\widehat{\mathsf{GC}}$ using encoding information $\widehat{e}$ cannot go back to both the wire label and the hash being correct.

1.4 Related Work

To our knowledge, there is no prior work specifically addressing hashing of GCs. At the same time, significant research effort has been expended on optimizing core GC performance. Work includes algorithmic GC improvements, such as Free XOR [KS08a], FleXOR [KMR14], half-gates [ZRE15], as well as optimizing underlying primitives, such as JustGarble [BHKR13]. Our work complements the existing GC improvement work.

Of course, the natural GC hashing approach works: just hash the generated GC. The problem with this is, of course, its cost. Relative cost of fixed-key cipher garbling and hashing are strongly architecture-dependent. They can be almost the same (e.g., when both AES and SHA are implemented in hardware). In another extreme, Intel's white paper [GGO$^+$] reports that AES-NI evaluation of 16-byte blocks is 23× faster that that of SHA1 ($35,965.9$ vs $793,718.7$ KB/sec). In our experiments reported in Section 5.2, we observed about 6× performance difference between AES-NI and SHA1.

Improving on this, and motivated in part by the availability of fast hardware AES implementations, there was a short series of works [BRS02,RS08b,RS08a,BÖS11], implementing a hash function with three fixed-key AES function calls. A recent work of Rogaway and Steinberger [RS08a] constructs a class of linearly-determined, permutation-based compression functions $\{0,1\}^{mn} \rightarrow \{0,1\}^{rn}$ making k calls to the different permutations π_i for $i \in [k]$, where they named their construction as LPmkr. The fastest construction LP362 (12.09 cycles per byte) [BÖS11], with 6 calls to fixed-key AES would cost about 6× of that of fast garbling. Davies-Meyer-based hash construction [Win84] in the ideal cipher model considered in literature is reported to have similar speeds [BÖS11].

In comparison, our work eliminates the cost of hash whatsoever, while adding no cost to garbling or GC evaluation.

C&C and uses of hashed GC. There is a long sequence of GC-based SFE work, e.g. [Lin13,HKE13,Bra13,LR14,HKK⁺14,KM15], most of which uses some form of C&C or challenging the GC generator. Based on [GMS08], these works will benefit from our result, to varying degree. The exact performance benefit will depend on where the Free Hash is used, the ratio of evaluated/test circuits, as well as the computational/communication resources available to the players. In Section 5, we calculate performance improvement in several C&C protocols due to our GC hash.

2 Preliminaries

Notation. Let PPT denote probabilistic polynomial time. We let λ be the security parameter, $[n]$ denote the set $\{1, ..., n\}$, and $|\boldsymbol{t}|$ denote the number of bits in a string. We denote the i-th bit value of a string $\boldsymbol{s}$ by $\boldsymbol{s}[i]$, use $||$ to denote concatenation of bit strings. We write $x \stackrel{R}{\leftarrow} \mathcal{X}$ to mean sampling a value x uniformly from the set $\mathcal{X}$. For a bit string $\boldsymbol{s}$, we let $\boldsymbol{s}^{\lll i}$ denote the bit string obtained by shifting $\boldsymbol{s}$ by i bits to the left. Throughout, by shift we mean a *circular* shift, where the vacant bit positions are filled not by zeros but by the shifted bits. $\mathsf{lsb}(\boldsymbol{s})$ denotes the least significant bit of string $\boldsymbol{s}$. We say a function $f(\cdot)$ is negligible if $\forall c \in \mathbb{N}$, there exists $n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$, it holds that $f(n) < n^{-c}$.

Let S be an infinite set and $X = \{X_s\}_{s \in S}, Y = \{Y_s\}_{s \in S}$ be distribution ensembles. We say X and Y are computationally indistinguishable, if for any PPT distinguisher $\mathcal{D}$ and all sufficiently large $s \in S$, we have $|\Pr[\mathcal{D}(X_s) = 1] - \Pr[\mathcal{D}(Y_s) = 1]| < 1/p(|s|)$ for every polynomial $p(\cdot)$.

Ideal cipher model. The Ideal Cipher Model (ICM) is an idealized model of computation, similar to the random oracle model (ROM) [BR93]. In ICM, one has a publicly accessible random block cipher (or ideal cipher). This is a block cipher with a k-bit key and a n-bit input/output, that is chosen uniformly at random among all block ciphers of this form; this is equivalent to having a family of 2^k independent random permutations. All parties including the adversary can make both encryption and decryption queries to the ideal block cipher, for any given key. ICM is shown to be equivalent to ROM [CPS08].

Collision-resistant hash function. A hash function family $\mathcal{H}$ is a collection of functions, where each $H \in \mathcal{H}$ is a mapping from $\{0,1\}^m$ to $\{0,1\}^n$, such that $m > n$ and m, n are polynomials in security parameter λ. An instance $H \in \mathcal{H}$

can be described by a key which is public known. We say a hash function family $\mathcal{H}$ is *collision-resistant* if for any PPT adversary $\mathcal{A}$

$$\mathbf{Pr}[H \stackrel{R}{\leftarrow} \mathcal{H}, (x, x') \leftarrow \mathcal{A}(H) : x \neq x' \wedge H(x) = H(x')] = \mathsf{negl}(\lambda)$$

2.1 Yao's construction

A comprehensive treatment of Yao's construction of garbled circuits, was given in [LP09]. At a high-level, in Yao's construction, each wire of the boolean circuit is associated with two random strings called wire labels or wire keys that encode logical 0 and 1 wire values. A garbled truth table is constructed for every gate in the circuit, where each combination of input wire labels is used to encrypt the appropriate output wire label as per the gate functionality. This results in four ciphertexts per gate, one for each input combination of the gate. The evaluator knows only one label for each input wire, and can therefore, open only one of the four ciphertexts.

2.2 Garbled Circuits

We make use of the abstraction of garbling schemes [BHR12] introduced by Bellare et al. At a high-level, a garbling scheme consists of the following algorithms: Gb takes a circuit as input and outputs a garbled circuit, encoding information, and decoding information. En takes an input x and encoding information and outputs a garbled input X. Eval takes a garbled circuit and garbled input X and outputs a garbled output Y. Finally, De takes a garbled output Y and decoding information and outputs a plain circuit-output (or an error $\perp$).

We note that this deviates from the definition of [BHR12], in that, we include $\perp$ in the range of the decoding algorithm De, so it now outputs a plain output value corresponding to a garbled output value or $\perp$ if the garbled output value is invalid. [JKO13] add an additional verification algorithm Ve to the garbling scheme. Formally, we define a *verifiable garbling scheme* by a tuple of functions $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Eval}, \mathsf{De}, \mathsf{Ve})$ with each function defined as follows.

- *Garbling* algorithm $\mathsf{Gb}(1^\lambda, \mathcal{C})$: A randomized algorithm which takes as input the security parameter and a circuit $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$ and outputs a tuple of strings $(\mathsf{GC}, \{X_j^0, X_j^1\}_{j\in[n]}, \{Z_j^0, Z_j^1\}_{j\in[m]})$, where GC is the garbled circuit, the values $\{X_j^0, X_j^1\}_{j\in[n]}$ denote the input-wire labels, and the values $\{Z_j^0, Z_j^1\}_{j\in[m]}$ denote the output-wire labels.
- *Encode* algorithm $\mathsf{En}(x, \{X_j^0, X_j^1\}_{j\in[n]})$: a deterministic algorithm that outputs the input wire labels $\mathbf{X} = \{X_i^{\boldsymbol{x}[i]}\}_{i\in[n]}$ corresponding to input x.
- *Evaluation* algorithm $\mathsf{Eval}(\mathsf{GC}, \{X_j\}_{j\in[n]})$: A deterministic algorithm which evaluates garbled circuit GC on input-wire labels $\{X_j\}_{j\in[n]}$, and outputs a garbled output $\mathbf{Y}$.
- *Decode* algorithm $\mathsf{De}(\mathbf{Y}, \{Z_j^0, Z_j^1\}_{j\in[m]})$: A deterministic algorithm that outputs the plaintext output corresponding to $\mathbf{Y}$ or $\perp$ signifying an error if the garbled output $\mathbf{Y}$ is invalid.

– *Verification* algorithm $\mathsf{Ve}(\mathcal{C}, \mathsf{GC}, \{Z_j^0, Z_j^1\}_{j\in[m]}, \{X_j^0, X_j^1\}_{j\in[n]})$: A deterministic algorithm which takes as input a circuit $\mathcal{C}$, garbled circuit GC, input-wire labels $\{X_j^0, X_j^1\}_{j\in[n]}$, and output-wire labels $\{Z_j^0, Z_j^1\}_{j\in[m]}$ and outputs accept if GC is a valid garbling of $\mathcal{C}$ and reject otherwise.

A verifiable garbling scheme may satisfy several properties such as *correctness, privacy, obliviousness, authenticity and verifiability.* We now review some of these notions: (1) *correctness*, (2) *privacy* (3) *authenticity*, and (4) *verifiability.* The definitions for correctness and authenticity are standard: correctness enforces that a correctly garbled circuit, when evaluated, outputs the correct output of the underlying circuit; authenticity enforces that the evaluator can only learn the output label that corresponds to the value of the function. *Verifiability* [JKO13] allows one to check that the garbled circuit indeed implements the specified plaintext circuit $\mathcal{C}$.

We include the definitions of these properties for completeness.

Definition 2.1 *(Correctness) A garbling scheme $\mathcal{G}$ is* $\mathsf{correct}$ *if for all input lengths $n \leq \mathsf{poly}(\lambda)$, circuits $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$ and inputs $x \in \{0,1\}^n$, the following probability is negligible in λ:*

$$\Pr(\mathsf{De}(\mathsf{Eval}(\mathsf{GC}, \{X_j^{x_j}\}_{j\in[n]}), \{Z_j^0, Z_j^1\}_{j\in[m]}) \neq \mathcal{C}(x) : \\ (\mathsf{GC}, \{X_j^0, X_j^1\}_{j\in[n]}, \{Z_j^0, Z_j^1\}_{j\in[m]}) \leftarrow \mathsf{Gb}(1^\lambda, \mathcal{C})).$$

Definition 2.2 *(Privacy) A garbling scheme $\mathcal{G}$ has* $\mathsf{privacy}$ *if for all input lengths $n \leq \mathsf{poly}(\lambda)$, circuits $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$, there exists a* PPT *simulator* Sim *such that for all inputs $x \in \{0,1\}^n$, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the following two distributions are computationally indistinguishable:*

– $\textsc{Real}(f, x)$ *: run* $(\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\lambda, \mathcal{C})$*, and output* $(\mathsf{GC}, \mathsf{En}(x, e), d)$.
– $\textsc{Ideal}_{\mathsf{Sim}}(\mathcal{C}, f(x))$*: output* $\mathsf{Sim}(1^\lambda, \mathcal{C}, \mathcal{C}(x))$

Definition 2.3 *(Authenticity) A garbling scheme $\mathcal{G}$ is* $\mathsf{authentic}$ *if for all input lengths $n \leq \mathsf{poly}(\lambda)$, circuits $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$, inputs $x \in \{0,1\}^n$, and all probabilistic polynomial-time adversaries $\mathcal{A}$, the following probability is negligible in λ:*

$$\Pr\left(\begin{array}{c} \widehat{Y} \neq \mathsf{Eval}(\mathsf{GC}, \{X_j^{x_j}\}_{j\in[n]}) \\ \wedge \mathsf{De}(\widehat{Y}, \{Z_j^0, Z_j^1\}_{j\in[m]}) \neq \perp \end{array} : \begin{array}{c} (\mathsf{GC}, \{X_j^0, X_j^1\}_{j\in[n]}, \{Z^0, Z^1\}_{j\in[m]}) \leftarrow \mathsf{Gb}(1^\lambda, \mathcal{C}) \\ \widehat{Y} \leftarrow \mathcal{A}(\mathcal{C}, x, \mathsf{GC}, \{X_j^{x_j}\}_{j\in[n]}) \end{array}\right).$$

Definition 2.4 *(Verifiability) A garbling scheme $\mathcal{G}$ is* $\mathsf{verifiable}$ *if for all input lengths $n \leq \mathsf{poly}(\lambda)$, circuits $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$, inputs $x \in \{0,1\}^n$, and all probabilistic polynomial-time adversaries $\mathcal{A}$, the following probability is negligible in λ:*

$$\Pr\left(\mathsf{De}(\mathsf{Eval}(\mathsf{GC}, \mathsf{En}(x, e)), d) \neq \mathcal{C}(x) : \begin{array}{c} (\mathsf{GC}, e, d) \leftarrow \mathcal{A}(1^\lambda, \mathcal{C}) \\ \mathsf{Ve}(\mathcal{C}, \mathsf{GC}, d, e) = \mathsf{accept} \end{array}\right)$$

In the definition of verifiability above, we give the decoding information explicitly to the verification algorithm since in our construction the garbled circuit includes only the garbled tables and not the decoding information. We note that a natural and efficient way to obtain a verifiable garbling scheme is to generate GC by using the output of a pseudorandom generator on a seed as the random tape for Gb, and then provide the seed to the verification procedure Ve. Ve will regenerate the GC and the encoding and decoding tables, and will output accept for a garbled circuit if and only if it is equal to the generated one.

2.3 Free-XOR and other optimizations

Several works have studied optimizations to reduce the size of a garbled gate down from four ciphertexts. Garbled row-reduction was introduced by Naor, Pinkas and Sumner [NPS99]. There, instead of choosing the wire labels at random for each wire, they are chosen such that the first ciphertext will be the all-zero string, and hence need not be sent. In [PSSW09], the authors describe a way to further reduce the number of ciphertexts per gate to 2, by applying polynomial interpolation at each gate. Kolesnikov and Schneider [KS08a] introduced the Free XOR approach, allowing evaluation of XOR gates without any cost. Here, the idea is to choose wire labels such that the two labels on the same wire have the same (secret) offset across the entire circuit. The two labels for a given wire are of the form $(A, A \oplus \Delta)$, where Δ is secret and common to all wires. Now, as first proposed in [Kol05], an evaluator who has one of $(A, A \oplus \Delta)$ and one of $(B, B \oplus \Delta)$ can compute the XOR by simply XORing the wire labels. The result is either C or $C \oplus \Delta$ where $C = A \oplus B$ and correctly represents the result of XOR. Thus, no ciphertexts are needed for the XOR gate. Kolesnikov, Mohassel and Rosulek proposed a generalization of Free XOR called FleXOR [KMR14]. In FleXOR, each XOR gate can be garbled using 0,1, or 2 ciphertexts, depending on certain structural properties of the circuit. In [ZRE15], the authors present a method built on Free XOR that can garble an AND gate using only two ciphertexts. This technique is also compatible with Free XOR. The idea is to write an AND gate as a combination of XOR and two *half-gates*, where a half-gate is an AND gate for which one party knows one of the inputs. The half-gates can be garbled with one ciphertext each, and the resulting AND gate, in combination with free-XOR, uses two ciphertexts.

3 Preliminary Discussion

3.1 Our Treatment of GC Topology and Formalization of the GC Representation

A formalization of what precisely the GC description string GC includes is often natural and hence is usually omitted from discussion. In our setting this an important aspect, as we focus on the collision resilience-related properties of GC strings, as well as on minimizing the size of GC and its computation time.

Firstly, we remind the reader that in the BHR [BHR12] notation the function Gb outputs the garbling *function* F. Since it is problematic to operate on functions, BHR regards Gb as operating on strings representing and defining the corresponding functions. In our notation, Gb outputs GC, which we treat as a string defining the evaluation process as well.

Clearly, GC will contain a set of garbled tables; the question is how to treat the circuit topology, i.e. exactly how to describe/define how Eval should process GC. One choice is to treat the plaintext circuit/topology as a part of GC. Because we focus on size/computation, this approach would cause some waste. Indeed, in most scenarios, the circuit and topology is known to both players, and hence could be implicit in GC.

Instead, we opt to consider the circuit description, including the locations of the free XOR gates as an externally generated string. It is certainly the case in SFE where the evaluated function is known to both players, and players can *a priori* adopt a convention on how to map the GC garbled gates to the circuit gates, hence defining the evaluation process. In PFE, which is the case in our certified function evaluation scenario (see Section 1.1), the evaluated function is *not* known to the evaluator. In this case, we still treat the topology/evaluation instructions as external to GC and assume that they are correctly delivered to the evaluator.

We note that in the certified function case, this can be naturally achieved by the CA signing the topology with a unique identifier, and including this identifier with GC and the hash of GC.

3.2 Our Assumptions

Our work optimizes high-performance primitives, and it is important to be clear on the assumptions we require of them so as to properly compare to related work.

We use the same primitives, and nearly identical constructions as JustGarble [BHKR13] and half-gates [ZRE15]. As a result, privacy and authenticity properties of our schemes hold under the same assumptions as [BHKR13,ZRE15], namely that the Davies-Meyer (DM) construction is a primitive meeting the guarantee of the random-permutation model (RPM). While [BHKR13] proves the security of their construction in RPM directly, [ZRE15] abstracts the DM security property as a variant of correlation-robust function. Our first (auxiliary) construction, namely, the privacy property, is proven under assumption that DM is correlation-robust.

To achieve hash security, we need to assume collision resistance of DM. We note that collision resistance of DM can be achieved e.g., by assuming that DM meets the requirement of the ideal-cipher model (ICM) [BRS02].

3.3 Cipher Instantiation

As noted above, we instantiate the key derivation function (KDF) calls as do [BHKR13,ZRE15], with the Davies-Meyer construction. Namely, the input

X to KDF $H(X, i)$ are the 128-bit long wire keys, and i is an internal integer that simply increments per hash function call. We set $H_\pi(X, i) = \pi(K) \oplus K$, where $K = 2x \oplus i$ (π is assumed to be an ideal cipher, instantiated with 128-bit AES with randomly chosen key).

3.4 Hash Security Parameters

We use $\lambda = 128$-bit security parameter, which is standard for encryption and GCs. However, 128-bit hash domain is often seen as insufficient. This is because of the birthday attack, which provides time-space tradeoff for an attacker. Specifically, a collision-finding attacker can precompute and store a square-root number of hash images. Then by birthday paradox, a random collision will be found among these images with significant probability. This attack requires 2^{64} hash computations and efficiently accessible storage for 2^{64} hash values.

We argue that 128-bit hash security is nevertheless acceptable in SFE, if used carefully.

Firstly, we note that computing 2^{64} hashes is an extremely expensive task. Indeed, recent Bitcoin reports [Bra] suggest that world's hashing power recently peaked at 1 PetaHash per second (i.e. $1000^5 < 2^{50}$ hashes/sec). That is, global Blockchain hashing power can compute 2^{64} hashes in the order of 2^{14} seconds (or 4.5 hours). Much more importantly, storage systems operate many orders of magnitude slower than CPUs and hashing ASICs, implying that storing and searching these hashes will take 10^3–10^6 times more time than generating them. Thus, extremely conservatively, we estimate that today a random hash collision may be found by engaging the *entire* Bitcoin mining system fitted with global-scale storage system in 4500 hours (about 6 months).

In the majority of applications, the time and financial expense to achieve such a task will not be feasible.

Importantly, SFE hash checks have an *online* property, meaning that we can set up the system such that preprocessing or post-processing will not aid the attacker. Indeed, consider the SFE scenario and the following solution. In the existing fixed-key cipher-based protocols it is specified that the fixed key is chosen at random prior to GC generation. We can simply explicitly require that *both* players contribute to key generation, and that the selected key will be the one defining the fixed-key permutation used in GC. This will render any precomputation useless. Post-computation, while a threat to the privacy and, perhaps, authenticity of GC, is not helping the attacker, since the GC evaluator decision to accept or reject reached during the execution, is irrevocable. GC evaluator can set a generous time limit (e.g. several seconds or even minutes) after which it will abort the execution. The probability of $\mathcal{A}$ cheating via finding a 128-bit hash collision in this period of time sufficiently small, even given entire world's resources available to $\mathcal{A}$.

In sum, we have argued that using 128-bit hash security is appropriate for SFE and the applications we discuss in this work. Further, as eventually we move from 128-bit AES to next-generation of ciphers, our hash security guarantee will benefit from the transition.

4 GC hashing scheme

In this section, we define our hashed garbled circuit scheme. We capture the security guarantees we require from this new notion, and then present our construction that outputs a garbled circuit and its hash. Our garbled circuit construction satisfies the properties of correctness, authenticity and privacy. We then show that our construction is secure according to our hash security definition.

4.1 Hashed Garbled Circuit security

Recall, we want to define hash security of garbled circuits with the same topology (cf. Section 3.1). We require that if the hash of such two garbled circuits collide, and one of them verifies correctly, then with high probability the other garbled circuit will fail evaluation. We now formalize this intuition in the definition below.

Definition 4.1 *(Hash security) A garbling scheme $\mathcal{G}$ is* hash-secure *with respect to a hash function $\mathcal{H}$ if for every boolean circuit $\mathcal{C}$, input x and* PPT *adversary $\mathcal{A}$,*

$$\Pr\left(\mathsf{De}(\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(x,\widehat{e})), d) \neq \perp \; : \; \begin{array}{c} \Big(\mathsf{GC}, \widehat{\mathsf{GC}}, e = \{X_j^0, X_j^1\}_{j\in[m]}, \\ \widehat{e} = \{\widehat{X}_j^0, \widehat{X}_j^1\}_{j\in[m]}, d, h\Big) \leftarrow \mathcal{A}(\mathcal{C}, 1^\lambda), \\ \mathsf{GC} \neq \widehat{\mathsf{GC}}, \\ \mathsf{Topology}(\mathsf{GC}) = \mathsf{Topology}(\widehat{\mathsf{GC}}), \\ \mathsf{Ve}(\mathcal{C}, \mathsf{GC}, d, e) = \mathsf{accept}, \\ \mathcal{H}(\mathsf{GC}) = \mathcal{H}(\widehat{\mathsf{GC}}) = h) \end{array}\right).$$

is negligible in λ.

We point out that the decoding information d that results in failed decoding of $\widehat{\mathsf{GC}}$ is the same decoding information with respect to which GC successfully verifies, and this is essential to hash security. If we did not place this requirement, then an adversary can change d to $\widehat{d}$ which decodes any string that Eval on $\widehat{\mathsf{GC}}$ returns. We note that in full generality it is not necessary to require $\mathcal{A}$ to generate a GC passing the verification Ve of a specific circuit $\mathcal{C}$. We can achieve that if an $\mathcal{A}$ generates two unequal GCs with the same hash, at least one of them will always output $\perp$. However, the above definition 4.1 reflects the typical use of GCs, and is sufficient for our construction.

In this work we consider verifiable garbling schemes with hash security. That is, $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Eval}, \mathsf{De}, \mathsf{Ve}, \mathcal{H})$. Because we apply our constructions to secure computation, we will need schemes additionally satisfying the properties of correctness (cf. Definition 2.1) and privacy (cf. Definition 2.2). If needed, the authenticity property of GC (cf. Definition 2.3) can be achieved as well.

4.2 Our Construction

We now formalize the intuition of Section 1.3 on how to generate a GC hash for free when garbling. The full construction is presented in Figure 1; here we provide additional intuition. Recall, in Section 1.3, we explained that after we generated (temporary) GC tables, we need to XOR their GT entries into the GC hash in one manner, and into the GC wire labels in another manner. In our construction, we do so by bitwise shifting the GT entries C_i prior to XORing them into the wire labels.

We note that we use bit shifting because it is fast and easy to implement, but a more general condition is sufficient for security of our scheme[4].

In presenting our construction, we adopt the approach used by [BHKR13] and others, where the gates are garbled as $H(w_i||w_j||r) \oplus w_k$, where w_i and w_j are wire labels on input wires, r is a nonce and w_k is a wire label on the output wire. H is a key-derivation function modeled as a random oracle.

The scheme we present below follows the standard point-and-permute optimization. This was introduced by Beaver, Micali and Rogaway in [BMR90], where a select bit is appended to each wire label, such that the two labels on each wire have opposite select bits. This association between select bits and the logical truth values is random and kept secret. Now the garbled truth table can be arranged by these public select bits. The evaluator can select the correct ciphertext to decrypt based on the select bit instead of trying all four. For each wire label w, its least significant bit $\mathsf{lsb}(w)$ is reserved as a select bit that is used as in the point-and-permute technique, and complementary wire labels have opposite select bits. For the ith wire, define $p_i = \mathsf{lsb}(w_i^0)$. When using Free XOR, the global randomly chosen offset R is such that $\mathsf{lsb}(R) = 1$. Since $w_i^0 \oplus w_i^1 = R$ holds for each i in the circuit, we have that $\mathsf{lsb}(w_i^0) \neq \mathsf{lsb}(w_i^1)$.

To simplify presentation, in our constructions and notation we set the decoding information simply to be the output wire labels. We note, this does not preserve the authenticity property of GC. Authenticity can be easily achieved in our scheme, e.g. by instead setting the decoding information to be the collision-resistant hashes of the output labels. In more detail, let H be a collision-resistant hash function. The output translation table for a wire will now be $\{H(w_i^0), H(w_i^1)\}$. Given a garbled value w_i^b on an output wire, it is possible to determine whether it corresponds to the 0 or 1 key by computing $H(w_i^b)$ and

[4] This condition is as follows. We set the wire labels of a gate output wire as a function of its temporary wire labels and the entries of the garbled gate table. Consider functions f_i such that, if

$$\bigoplus_{i=1}^{4} C_i = \bigoplus_{i=1}^{4} \widehat{C}_i$$

for $C_i \neq \widehat{C}_i$. Then,

$$\Pr[\bigoplus_{i=1}^{4} f_i(C_i) = \bigoplus_{i=1}^{4} f_i(\widehat{C}_i)]$$

is negligible. As we will later see in the proof, this is the property that we use in proving the hash security of our construction in proof of Theorem 4.6.

checking whether it is equal to the first or second value in the pair. However, given this output translation table, it is not feasible to find the actual garbled values.

Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a function, satisfying properties discussed in Section 3.2. For a function represented by a circuit $\mathcal{C} : \{0,1\}^n \to \{0,1\}^m$, we use $W_{\mathsf{in}}, W_{\mathsf{out}}$ to denote the input and output wires of f respectively, and G_{inter} for intermediate gates. The Free Hash garbling scheme $\mathsf{h}\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Eval}, \mathsf{Ve}, \mathcal{H})$ is described in Figure 1.

The construction in Figure 1 satisfies the properties of authenticity (cf. Definition 2.3), privacy (cf. Definition 2.2) and hash security (cf. Definition 4.1).

Theorem 4.2 *The Free Hash garbling scheme* $\mathsf{h}\mathcal{G}$ *described in Figure 1 satisfies privacy as in Definition 2.2 assuming the correlation robustness of* H*.*

Theorem 4.3 *The Free Hash garbling scheme* $\mathsf{h}\mathcal{G}$ *described in Figure 1 satisfies authenticity as in Definition 2.3 assuming the correlation robustness of* H*.*

We omit the proofs of privacy and authenticity in the main body, since our changes to the standard construction do not affect them, and closely follow the arguments of [BHKR13] and [ZRE15]. We include the proofs in the full version.

Hash security. We now state and prove a technical lemma on which we rely for proving hash security (Theorem 4.6). The lemma below captures the following useful fact about GC and $\widehat{\mathsf{GC}}$: a gate in $\widehat{\mathsf{GC}}$ whose $\mathsf{pad}_{i,2}$ (XOR hash of the gate table) collides with that of the gate in GC will not be evaluated correctly (i.e. will not produce a valid label on the output wire) if the gate table is different, or if the input wire keys of the gate are different, or both. We say that a wire label, obtained during evaluation on input x encoded using $\widehat{e}$, is valid if it is one of the two possible wire labels for the same wire in GC. For presentation, we slightly abuse notation, by writing g_i to mean both the gate and the garbled table corresponding to the gate. It will be clear from context, which of the two is meant.

Definition 4.4 *(Valid key) Let* $(\mathsf{GC}, e, \widehat{\mathsf{GC}}, \widehat{e}, d, h)$ *be such that* $\mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathsf{Topology}(\mathsf{GC}) = \mathsf{Topology}(\widehat{\mathsf{GC}}), \mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}) = h$ *and* $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$*. An internal wire key* $\widehat{K}_i^b$ *obtained on wire* w_i *during* Eval *of* $\widehat{\mathsf{GC}}$ *is called* valid *if* $\widehat{K}_i^b \in \{K_i^0, K_i^1\}$ *where* (K_i^0, K_i^1) *are the wire keys corresponding to* 0 *and* 1 *on wire* w_i *in* GC*.*

Lemma 4.5 *Let* $(\mathsf{GC}, e, \widehat{\mathsf{GC}}, \widehat{e}, d, h) \leftarrow \mathcal{A}(1^\lambda)$ *be such that* $\mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathsf{Topology}(\mathsf{GC}) = \mathsf{Topology}(\widehat{\mathsf{GC}}), \mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}) = h$ *and* $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$*. Assuming* $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$*, evaluation of the garbled gate* $\widehat{g}_i$ *during* Eval *results in a valid wire label for the output wire of the gate with probability* $\mathsf{negl}(\lambda)$ *in the following cases:*

1. *Input wire keys to gate* $\widehat{g}_i$ *are valid, and* $\widehat{g}_i \neq g_i$*.*
2. *At least one input wire key to gate* $\widehat{g}_i$ *is invalid and* $\widehat{g}_i = g_i$*.*

– $\mathsf{Gb}(1^\lambda, \mathcal{C})$: On input the security parameter λ and a circuit $\mathcal{C}$, choose $R \leftarrow \{0,1\}^{\lambda-1}||1$ and set $h = 0$.
  1. For each input wire $W_i \in W_{\mathsf{in}}$ of the circuit $\mathcal{C}$, set garbled labels in the following way: Randomly choose $K_i^0 \in \{0,1\}^\lambda$. Set $K_i^1 = K_i^0 \oplus R$. Set the garbled labels for input wire W_i as $w_i = (K_i^0, K_i^1)$.
  2. For each intermediate gate $G_i : W_c = g_i(W_a, W_b)$ of $\mathcal{C}$ in topological order:
     (a) Parse the garbled input labels as $w_a = (K_a^0, K_a^1)$ and $w_b = (K_b^0, K_b^1)$.
     (b) If G_i is an XOR gate, set garbled labels for the gate output wire W_c as $K_c^0 = K_a^0 \oplus K_b^0$, and $K_c^1 = K_c^0 \oplus R$.
     (c) If G_i is an AND gate
        - Choose temporary garbled labels for the gate output wire W_c as $T_c^0 \in \{0,1\}^\lambda$, and set $T_c^1 = T_c^0 \oplus R$.
        - Create G_i's garbled table: For each possible combination of G_i's input values $v_a, v_b \in \{0,1\}$, set $\tau^i_{v_a,v_b} = H(K_a^{v_a}|K_b^{v_b}|i) \oplus T_c^{g_i(v_a,v_b)}$. Sort entries τ^i in the table by input pointers, and let the entries be $C_{i,1}, C_{i,2}, C_{i,3}, C_{i,4}$.
        - For $d \in \{0,1\}$, compute:
$$\mathsf{pad}_{i,1} = C_{i,1}^{\lll 1} \oplus C_{i,2}^{\lll 2} \oplus C_{i,3}^{\lll 3} \oplus C_{i,4}^{\lll 4}$$
$$K_c^0 = T_c^0 \oplus \mathsf{pad}_{i,1}$$
          Set the garbled labels for wire W_c as
$$w_c = (K_c^0, K_c^1), \text{ where } K_c^1 = K_c^0 \oplus R$$
        - Define
$$\mathsf{pad}_{i,2} = C_{i,1} \oplus C_{i,2} \oplus C_{i,3} \oplus C_{i,4}$$
$$h = h \oplus \mathsf{pad}_{i,2}$$
  3. For each output wire $W_i \in W_{\mathsf{out}}$ of $\mathcal{C}$, set $d_i^0 = (0, K_i^0)$ and $d_i^1 = (1, K_i^1)$
  4. Output encoding information e, decoding information d, garbled circuit GC and hash $\mathcal{H}(\mathsf{GC})$ as
$$e = \{(K_i^0, K_i^1)\}_{W_i \in W_{\mathsf{in}}}, d = \{(d_i^0, d_i^1)\}_{W_i \in W_{\mathsf{out}}}, \mathsf{GC} = \{\tau^i_{a,b}\}_{\substack{a,b\in\{0,1\}\\ G_i \in G_{\mathsf{inter}}}}, \mathcal{H}(\mathsf{GC}) = h$$

– $\mathsf{En}(\boldsymbol{x}, e)$: On input encoding information e and input $\boldsymbol{x}$, output encoding $\mathbf{X} = \{X_i^{\boldsymbol{x}[i]}\}_{i\in[n]}$.
– $\mathsf{De}(\mathbf{Y}, d)$: On input the decoding information d and the garbled output of the circuit $\mathbf{Y} = (Y_1, ..., Y_m)$, for each output wire i of the circuit $\mathcal{C}$, parse d as $d = \{(0, K_i^0), (1, K_i^1)\}_{i\in[m]}$. Then, set $y_i = b$ if $Y_i = K_i^b$ and $y_i = \perp$ if $Y_i \notin \{K_i^0, K_i^1\}$. Output the result $\boldsymbol{y} = (y_1, ..., y_m)$ if $\forall i, y_i \neq \perp$. Else, output $\perp$.
– $\mathsf{Eval}(\mathsf{GC}, \mathbf{X})$: On input the garbled circuit GC and garbled input $\mathbf{X}$, for each gate $G_i : W_c = g_i(W_a, W_b)$ with garbled inputs $w_a = K_a^{v_a}, w_b = K_b^{v_b}$. If G_i is an XOR gate, compute $w_c^{g_i(v_a,v_b)} = K_a^{v_a} \oplus K_b^{v_b}$. If G_i is an AND gate:
  1. Let C_1, C_2, C_3, C_4 be the table entries. Compute $\mathsf{pad} = \bigoplus_{i=1}^{4} C_i^{\lll i}$.
  2. Decode the temporary output value from garbled table entry τ^i in position (v_a, v_b) as $T_c^{g_i(v_a,v_b)} = H(K_a^{v_a}|K_b^{v_b}|i) \oplus \tau^i$.
  3. Compute the garbled value as $w_c^{g_i(v_a,v_b)} = T_c^{g_i(v_a,v_b)} \oplus \mathsf{pad}$.

– $\mathsf{Ve}(\mathcal{C}, \mathsf{GC}, d, e)$: Check that each gate in GC correctly encrypts the gate in $\mathcal{C}$ given the encoding information e. If yes, then output accept, else output reject.
– $\mathcal{H}(GC)$: On input the garbled circuit GC, output h as the XOR of all ciphertexts,
$$h = \bigoplus_{g_i} (C_{i,1} \oplus C_{i,2} \oplus C_{i,3} \oplus C_{i,4})$$

Fig. 1: The Free Hash garbling scheme $\mathsf{h}\mathcal{G}$

3. *At least one input wire key to gate* $\widehat{g_i}$ *is invalid, and* $\widehat{g_i} \neq g_i$.

Proof. Let $g_i = \{C_1, C_2, C_3, C_4\}$ be the ith garbled table in GC and $\widehat{g_i} = \{\widehat{C_1}, \widehat{C_2}, \widehat{C_3}, \widehat{C_4}\}$ the ith garbled table in $\widehat{\mathsf{GC}}$.

Case 1 Since $\widehat{g_i} \neq g_i$, w.l.o.g., let $C_1 \neq \widehat{C_1}$. Since $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$, there must be (at least) one $j \neq 1$ such that $\widehat{C_j} \neq C_j$. Now, $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$ gives,

$$\widehat{C_j} \oplus \widehat{C_1} = C_j \oplus C_1 \tag{1}$$

Let $\widehat{K} = (\widehat{K}_a, \widehat{K}_b)$ be the input wire key to gate g_i in $\widehat{\mathsf{GC}}$ during Eval, which by assumption is valid.
For the sake of contradiction, say, one of the ciphertexts, say, $\widehat{C_1}$, in $\widehat{g_i}$ gives a valid output wire key. Let T be the intermediate key obtained by decrypting $\widehat{C_1}$. Now validity of output wire key implies $T \oplus \widehat{\mathsf{pad}}_{i,1} = K \in \{K^0, K^1\}$.

$$\begin{aligned} T \oplus \widehat{\mathsf{pad}}_{i,1} &= K \\ \widehat{C_j}^{\lll j} \oplus \widehat{C_1}^{\lll 1} &= C_j^{\lll j} \oplus C_1^{\lll 1} \oplus R \end{aligned} \tag{2}$$

where $R = T \oplus K \oplus \mathsf{pad}_{i,1}$ is a fixed value, and $T = H(\widehat{K}||i) \oplus \widehat{C_1}$. Therefore, K is valid only when both (1) and (2) hold. We now argue that this happens with probability $\leq 1/2^\lambda$. By the assumption that $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$, C_1 and C_j are random keys masked by the outputs of the function H. If, therefore, a $\widehat{C_1}$ and $\widehat{C_j}$ that satisfies (1), also satisfies (2), then we can find r_1 and r_2 such that $r_1 \oplus r_2$ is δ for some fixed δ and $r_1^{\lll} \oplus r_2^{\lll}$ collides with the output of the function H on a fixed value. By collision resistance of the function H, this happens only with probability $\leq 1/2^\lambda$.

Case 2 $g_i = \widehat{g_i}$. Either $\widehat{K}_a \notin \{K_a^0, K_a^1)$ or $\widehat{K}_b \notin \{K_b^0, K_b^1)$ or both, where (K_a^0, K_a^1) and (K_b^0, K_b^1) are the wire keys corresponding to the input wires of $\widehat{g_i}$ in GC. Let (K^0, K^1) be the wire keys of the output wire of g_i.
For the sake of contradiction, say, one of the ciphertexts, say, C_1, gives a valid output wire key with $\widehat{K}$ as the input wire keys. Let T be the intermediate key obtained by decrypting C_1. Now validity of output wire key implies $T \oplus \mathsf{pad}_{i,1} = K \in \{K^0, K^1\}$. That is,

$$H(\widehat{K}||i) \oplus C_1 \oplus \mathsf{pad}_{i,1} = K \tag{3}$$

K is valid when (3) holds, and that happens with negligible probability since we can find a r such that the output of H on r collides with a given value only with probability $\leq 1/2^\lambda$.

Case 3 W.l.o.g., let $C_1 \neq \widehat{C_1}$. Since $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$, there must be (at least) one $j \neq 1$ such that $\widehat{C_j} \neq C_j$.
Either $\widehat{K}_a \notin \{K_a^0, K_a^1)$ or $\widehat{K}_b \notin \{K_b^0, K_b^1)$ or both, where (K_a^0, K_a^1) and $(K_b^0 K_b^1)$ are the wire keys corresponding to the input wires of $\widehat{g_i}$ in GC. (K^0, K^1) be the wire keys of the output wire of g_i.

Now, $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$ gives,

$$\widehat{C_j} \oplus \widehat{C_1} = C_j \oplus C_1 \tag{4}$$

Let $\widehat{K} = (\widehat{K}_a, \widehat{K}_b)$ be the input wire key to gate g_i in $\widehat{\mathsf{GC}}$ during Eval. Since $\widehat{K}$ is invalid by assumption, either $\widehat{K}_a \notin \{K_a^0, K_a^1)$ or $\widehat{K}_b \notin \{K_b^0, K_b^1)$ or both, where (K_a^0, K_a^1) and (K_b^0, K_b^1) are the wire keys corresponding to the input wires of $\widehat{g}_i$ in GC. (K^0, K^1) be the wire keys of the output wire of g_i. For the sake of contradiction, say, one of the ciphertexts, say, $\widehat{C_1}$, in $\widehat{g}_i$ gives a valid output wire key. Let T be the intermediate key obtained by decrypting $\widehat{C_1}$. Now validity of output wire key implies $T \oplus \widehat{\mathsf{pad}}_{i,1} = K \in \{K^0, K^1\}$.

$$\begin{aligned} T \oplus \widehat{\mathsf{pad}}_{i,1} &= K \\ \widehat{C_j}^{\lll j} \oplus \widehat{C_1}^{\lll 1} &= C_j^{\lll j} \oplus C_1^{\lll 1} \oplus R \end{aligned} \tag{5}$$

where $R = T \oplus K \oplus \mathsf{pad}_1$, and $T = H(\widehat{K}||i) \oplus \widehat{C_1}$. Therefore, K is valid only when both (4) and (5) hold. We now argue that this happens with probability $\leq 1/2^\lambda$. By the assumption that $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$, C_1 and C_j are random keys masked by the outputs of the function H. If, therefore, $\widehat{K}$, $\widehat{C_1}$ and $\widehat{C_j}$ satisfy (4) and (5), then we can find r, r_1 and r_2 such that the output of the function H on r collides with $r_1^{\lll} \oplus r_2^{\lll}$ and $r_1 \oplus r_2$ is δ for some fixed δ. By collision resistance of the function H, this happens with probability at most $1/2^\lambda$.

When there is more than one $j \neq 1$ such that $\widehat{C_j} \neq C_j$ in cases (1) and (3) above, we will have,

$$\begin{aligned} \bigoplus_{j \neq 1} \widehat{C_j} \oplus \widehat{C_1} &= \bigoplus_{j \neq 1} C_j \oplus C_1 \\ \bigoplus_{j \neq 1} \widehat{C_j}^{\lll j} \oplus \widehat{C_1}^{\lll 1} &= \bigoplus_{j \neq 1} C_j^{\lll j} \oplus C_1^{\lll 1} \oplus R \end{aligned}$$

and the same arguments extend.

□

Theorem 4.6 *The Free Hash garbling scheme* $\mathsf{h}\mathcal{G}$ *described in Figure 1 satisfies hash security as defined in Definition 4.1 assuming the collision-resistance of* H*.*

Proof. Given an adversary $\mathcal{A}$ who outputs $(\mathsf{GC}, e, \widehat{\mathsf{GC}}, \widehat{e}, d, h)$ such that $\mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}) = h$, $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$, we show that $\forall x, \Pr[\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(x, \widehat{e})) \neq \perp] = \mathsf{negl}(\lambda)$. Since $\mathsf{GC} \neq \widehat{\mathsf{GC}}$, they differ in at least one garbled gate. Let g_i be the first gate in topological order that differs in GC and $\widehat{\mathsf{GC}}$. When $\mathsf{pad}_{i,2} = \widehat{\mathsf{pad}}_{i,2}$ for all $\widehat{g}_i \neq g_i$, by case (1) of Lemma 4.5, we have that the output wire key for $\widehat{g}_i$ is invalid. Now, by inductively applying cases

(2) and (3) of Lemma 4.5, all wire keys from then on, in topological order of evaluation remain invalid.

Now, when $\mathsf{pad}_{i,2} \neq \widehat{\mathsf{pad}}_{i,2}$, Eval on $\widehat{\mathsf{GC}}$ can return to a valid wire key for the output wire of $\widehat{g}_i \neq g_i$. Let us denote by $\widehat{\mathcal{H}}_i$ the running hash up until gate $\widehat{g}_i$ in $\widehat{\mathsf{GC}}$. Since $\mathsf{pad}_{i,2} \neq \widehat{\mathsf{pad}}_{i,2}$, we have $\widehat{\mathcal{H}}_i \neq \mathcal{H}_i$. By the assumption that $\mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC})$, there must be a gate $\widehat{g}_j \neq g_j$ such that

$$\varDelta = \bigoplus_{i:\mathsf{pad}_{i,2} \neq \widehat{\mathsf{pad}}_{i,2}} (\mathsf{pad}_{i,2} \oplus \widehat{\mathsf{pad}}_{i,2}) = \widehat{\mathcal{H}}_i \oplus \mathcal{H}_i \tag{6}$$

$$\widehat{\mathsf{pad}}_{j,2} = \mathsf{pad}_{j,2} \oplus \varDelta \tag{7}$$

We now argue that the output wire of $\widehat{g}_j$ is invalid. From an argument similar to case (1) of Lemma 4.5 (since the input wire keys to $\widehat{g}_j$ are valid), (7) imposes a constraint on the ciphertexts of $\widehat{g}_j$. Thus the probability that output wire key is valid is bounded by the probability of finding r_1 and r_2 such that $r_1 \oplus r_2$ is δ for some fixed δ and $r_1^{\lll} \oplus r_2^{\lll}$ collides with the output of the function H on a fixed value. By collision resistance of the function H, this happens only with probability $\leq 1/2^{\lambda}$.

By Lemma 4.5 and the union bound, we have that, $\Pr[\mathsf{De}(\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(x, \widehat{e})), d) \neq \bot] \leq |C|q^2/2^{\lambda}$, where $|C|$ is the number of gates in the circuit, and q is the number of queries to the function H that $\mathcal{A}$ is allowed to make.

Since the input x that lead to the above wire labels was arbitrary, we have that, given $\mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}), \mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$,

$$\forall x, \Pr[\mathsf{De}(\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(x, \widehat{e})), d) \neq \bot] = \mathsf{negl}(\lambda)$$

□

As calculated in the proof, the probability of hash collision is bounded by $|C|q^2/2^{\lambda}$. See Section 3.4 for discussion on parameter choices.

4.3 Hashing in half-gates garbling scheme

The current state of the art for garbled circuit construction is the half-gates scheme of Zahur et al. In the half-gates construction, two ciphertexts are used for each AND gate and the construction is compatible with the free-XOR technique [KS08a]. A half-gate is a garbled AND gate where one of the inputs to the gate is known in clear to one of the parties. Consider an AND gate $c = a \wedge b$. Now suppose the generator chooses a uniformly random bit r, and imagine we can have the evaluator learn the value of $r \oplus b$. We can write c as

$$c = a \wedge b = (a \wedge r) \oplus (a \wedge (r \oplus b))$$

[ZRE15] show how to garble the first AND gate with a generator-half-gate where the generator knows one of the values r, and the second AND gate with evaluator-half-gate since the evaluator know $r \oplus b$. The full AND gate is garbled by taking XOR of the two half-gates. Each garbled half-gate is one ciphertext, and with free-XOR, the full AND gate is two ciphertexts.

Let $\mathsf{GC}' = (\mathsf{Gb}', \mathsf{En}', \mathsf{De}', \mathsf{Eval}')$ be the algorithms of the half-gate garbling procedure in [ZRE15]. The algorithms for encoding and evaluation in our scheme are the same; we only include the garbling and decoding algorithms, Gb and De. We assume that the half-gate garbling scheme outputs wire labels corresponding to both 0 and 1 on the output wires as the decoding information. Gb outputs a garbled circuit, the encoding and decoding information and the hash of the garbled circuit. De returns a decoded output or $\perp$ if the garbled output is invalid.

- $\mathsf{Gb}(1^\lambda, \mathcal{C})$: On input security parameter λ and a circuit $\mathcal{C}$, run the half-gate garbling algorithm $(e', d', \mathsf{GC}') \leftarrow \mathsf{Gb}'(1^\lambda, \mathcal{C})$, where $\mathsf{GC}' = \{\tau_{G_i}, \tau_{E_i}\}_{g_i \in G_{\mathsf{inter}}}$, and $d' = \{(d_i^0, d_i^1)\}_{W_i \in W_{\mathsf{out}}}$. Set the encoding information e, decoding information d, garbled circuit GC and hash $\mathcal{H}(\mathsf{GC})$ as

$$e = e', \quad d = d', \quad \mathsf{GC} = \mathsf{GC}', \quad \mathcal{H}(\mathsf{GC}) = \bigoplus_i (\tau_{G_i} \oplus \tau_{E_i})$$

- En is defined to be En' .
- Eval is defined to be Eval'.
- $\mathsf{De}(\mathbf{Y}, d)$: On input the decoding information d and the garbled output of the circuit $\mathbf{Y} = (Y_1, ..., Y_m)$, for each output wire i of the circuit $\mathcal{C}$, parse d as $d = \{d_i^0, d_i^1\}_{i \in [m]}$. Then, set $y_i = b$ if $Y_i = d_i^b$ and $y_i = \perp$ if $Y_i \notin \{d_i^0, d_i^1\}$. Output the result $\boldsymbol{y} = (y_1, ..., y_m)$ if $\forall i, y_i \neq \perp$. Else, output $\perp$.

Fig. 2: The Half-Gate Free Hash garbling scheme $\mathsf{half}\mathcal{G}$

Note that in the construction of hashed garbling scheme for half-gates above, the hash is the XOR of all the ciphertexts. Unlike our construction for general garbled circuits (cf. Figure 1), we do not modify the wire keys. Since the garbled circuit is the same as the original half-gates construction, we retain the privacy and authenticity properties. To argue hash security, first observe that in the half-gates scheme both ciphertexts in a garbled gate (one per half-gate) are decrypted and used for output wire computation. Consider an attacker $\mathcal{A}$ which modifies a gate table and changes one entry to decrypt to a wrong label. Then there must be another modified entry to correct the hash, and *both* modified entries need to decrypt correctly during evaluation to produce a valid label. Thus, in the half-gate garbling, the intuition for hash security is similar to that of our original 4-row construction. Namely, any modified gate will break the XOR hash. Further, any gate table that brings back the hash to the correct value will result in an invalid output wire label. We provide a proof sketch below.

Theorem 4.7 *The Half-Gate Free Hash garbling scheme* $\mathsf{half}\mathcal{G}$ *described in Figure 2 satisfies hash security as defined in Definition 4.1 assuming the collision-resistance of* H*.*

Proof Sketch. Given an adversary $\mathcal{A}$ who outputs $(\mathsf{GC}, e, \widehat{\mathsf{GC}}, \widehat{e}, d, h)$ such that $\mathsf{GC} \neq \widehat{\mathsf{GC}}, \mathcal{H}(\widehat{\mathsf{GC}}) = \mathcal{H}(\mathsf{GC}) = h$, $\mathsf{Ve}(\mathsf{GC}, d, e) = \mathsf{accept}$, we show that $\forall x, \Pr[\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(\widehat{e}, x)) \neq \perp] = \mathsf{negl}(\lambda)$. Since $\mathsf{GC} \neq \widehat{\mathsf{GC}}$, they must differ in at least one garbled gate, and let $g_i \neq \widehat{g}_i$ be the first gate in topological order that differs: $g_i = \{\tau_{G_i}, \tau_{E_i}\}$ and $\widehat{g}_i = \{\widehat{\tau}_{G_i}, \widehat{\tau}_{E_i}\}$. Let $\widehat{\mathcal{H}}_i$ be the running hash up until gate $\widehat{g}_i$ in $\widehat{\mathsf{GC}}$. We consider the following cases:

1. $\widehat{\mathcal{H}}_i = \mathcal{H}_i$ where $\mathcal{H}_i$ is the running hash until gate g_i in GC. Now $g_i \neq \widehat{g}_i$ and $\widehat{\mathcal{H}}_i = \mathcal{H}_i$ implies that both half-gates are modified since $\widehat{g}_i$ is the first gate that differs from GC. That is,

   $$\tau_{G_i} \neq \widehat{\tau}_{G_i} \text{ and } \tau_{E_i} \neq \widehat{\tau}_{E_i}$$

   Let $(\widehat{K}_a, \widehat{K}_b)$ be the input wire keys of $\widehat{g}_i$. The output wire key of $\widehat{g}_i$ during Eval is given by

   $$\widehat{K} = H(\widehat{K}_a) \oplus s_a \widehat{\tau}_{G_i} \oplus H(\widehat{K}_b) \oplus s_b(\widehat{\tau}_{E_i} \oplus \widehat{K}_a)$$

   where s_a and s_b are select bits. The probability that $\widehat{K}$ is valid is at most $1/2^\lambda$ by the collision resistance of function H. Now, by inductively using argument similar to cases (2) and (3) of Lemma 4.5, the wire keys of $\widehat{\mathsf{GC}}$ remain invalid.
2. $g_i \neq \widehat{g}_i$, $\widehat{\mathcal{H}}_i \neq \mathcal{H}_i$ and $\mathcal{H}(\mathsf{GC}) = \mathcal{H}(\widehat{\mathsf{GC}})$ implies there must be a gate $\widehat{g}_j \neq g_j$ such that

   $$\widehat{\tau}_{G_j} \oplus \widehat{\tau}_{E_j} = \widehat{\mathcal{H}}_i \oplus \mathcal{H}_i \oplus (\tau_{G_j} \oplus \tau_{E_j}) \quad (8)$$

   We now argue that the output wire of $\widehat{g}_j$ is invalid: (8) imposes a constraint on the ciphertexts of $\widehat{g}_j$. Thus the probability that output wire key is valid is bounded by the probability of finding r_1 and r_2 such that $r_1 \oplus r_2$ is δ for some fixed δ and r_1 and r_2 collide with the outputs of function H. By collision resistance of H, this happens with probability at most $1/2^\lambda$. Again, inductively all further wire keys of $\widehat{\mathsf{GC}}$ remain invalid.

By the union bound, we have that, $\Pr[\mathsf{De}(\mathsf{Eval}(\widehat{\mathsf{GC}}, \mathsf{En}(\widehat{e}, x)), d) \neq \perp] \leq |C|q^2/2^\lambda$, where $|C|$ is the number of gates in the circuit, and q is the number of queries to the function H that $\mathcal{A}$ is allowed to make. □

As calculated in the proof, the probability of hash collision is bounded by $|C|q^2/2^\lambda$. See Section 3.4 for discussion on parameter choices.

5 Performance and Impact

5.1 Cut-and-choose protocols using $\mathsf{h}\mathcal{G}$

As pointed out in [GMS08], an improvement in communication complexity can be achieved by taking the following approach. To compute a garbled circuit, the garbler P_1 generates a random PRG seed. Then the output of the pseudorandom generator is used as the random tape for the garbling algorithm. In C&C, P_1 sends to P_2 only a collision-resistant (CR) hash of each GC. In a later stage of the protocol, if a GC GC is chosen as a check circuit and needs to be opened, P_1 simply sends the seed corresponding to that circuit to P_2.

$\mathsf{h}\mathcal{G}$ hash can be used in C&C similarly to standard CR hash of GC. In [GMS08], P_1 commits via a collision resistant hash function to garbled circuits. These GCs can be either *good* or *cheating*. Importantly, due to the CR property of the hash, a malicious P_1 cannot change this designation at a later time. In using $\mathsf{h}\mathcal{G}$, P_1 has the same choice: he can compute $\mathsf{h}\mathcal{G}$ of either a good or a cheating GC. If he computed and sent the hash h of a good garbled circuit GC, then h cannot be claimed to match a cheating evaluation circuit $\widehat{\mathsf{GC}}$, even if the XOR hash $H(\mathsf{GC}) = H(\widehat{\mathsf{GC}})$. Indeed, w.h.p., evaluation of such a $\widehat{\mathsf{GC}}$ will fail and P_2 will abort, *independently* of P_2's input. Similarly, if P_1 computed and sent the hash of a cheating circuit $\widehat{\mathsf{GC}}$, it cannot be later opened as a good check circuit GC.

We stress that we must be careful when P_2 is allowed to abort, so as to not allow a selective failure attack. Specifically, a malicious P_1 could cause evaluation failure by sending an invalid label on a specific input wire/value pair or by generating a GC which produces an invalid label based on a value of an internal wire. Thus, while it is OK for P_2 to abort if it sees a GC which does not match the $\mathsf{h}\mathcal{G}$-hash, it should not (necessarily) abort simply based on seeing a decoding failure. Instead, this failure should be treated by the C&C procedure. We stress that it is protocol dependent, and protocol security should be evaluated. At the high level, our hashing guarantees that the garbler cannot open/equivocate an "honest" hashed circuit as a valid "malicious" circuit (or vice versa). However, he can open any (i.e. honest or malicious) hashed circuit as a "broken" one (i.e. one which will fail evaluation).

Covert C&C protocols [AL07,KM15], as well as C&C based on majority output, such as [LP11], can be made to work with $\mathsf{h}\mathcal{G}$. Indeed, exercising the extra power the adversary has (turning a good or bad evaluation circuit into a broken evaluation circuit) will simply cause covert evaluator to abort independently of its input. Similarly, in [LP11], the evaluation circuits which were made broken cannot be used to contribute to majority output. Using $\mathsf{h}\mathcal{G}$ with [KM15] requires a bit of care. [KM15] actually already explicitly support using [GMS08]. Using $\mathsf{h}\mathcal{G}$ differs from [GMS08] only in that a cheating P_1 can open an honest evaluation circuit as a broken one, resulting in an abort. However, the same effect could be achieved by P_1 sending an invalid signature on the garbled circuit.

We note that [Lin13] uses [LP11] as a basic step in cheating punishment and our $\mathsf{h}\mathcal{G}$ can be used within the [LP11] subprotocol of [Lin13]. However, it is not immediately clear $\mathsf{h}\mathcal{G}$ can be used elsewhere in [Lin13]. This is because the cheating punishment relies on evaluator having received a good evaluation circuit to recover the cheating garbler's input. However, in our case, malicious garbler can present a broken circuit, preventing input recovery.

Similarly, it is not immediately clear that the dual-execution C&C protocols of [HKE13,KMRR15] can take advantage of $\mathsf{h}\mathcal{G}$. Intuitively, this is because a malicious generator P_1 might produce a single cheating circuit, which is likely to be chosen for evaluation among a number of honest circuits. Then, P_1 will open all honest evaluation circuits as broken ones. Avoiding selective failure attack, P_2 will not abort, and the resulting output will depend on the output of the cheating circuit.

5.2 Implementation

We implemented our scheme using libgarble [Mal] for garbling and report on the performance below. In Table 1, we compare the cost of our GC hashing construction with garbling and then hashing the GC using SHA. We use the AES circuit to garble in the comparisons. The numbers in Table 1 are in cycles per gate. The configuration of the machine we used to run our implementation is: 2.3 GHz Core i5-2410M processor with 4 GB RAM. The processor has AES-NI integrated.

	Our $\mathsf{h}\mathcal{G}$ construction	Garble + SHA	justGarble
Standard Garbling	31.1	226.7	29
Half-gates	26.8	157.7	25.3

Table 1

We believe that free hashing will simplify and speed up GC use particularly in larger systems using GC, such as the Blind Seer encrypted database [PKV$^+$14,FVK$^+$15], where GC processing will be competing for the CPU resource with a number of other tasks.

SFE of private certified functions. We now consider the use case described in Section 1.1, where a Certificate Authority (CA) generates and certifies a number of GCs for use by the subscribers of the CA. In this case, clearly, CA is the bottleneck; Table 1 demonstrates over $6\times$ performance improvement for the state-of-the-art half-gates GC, as compared with using standard hashing available with the OpenSSL library. Again, we stress that with half-gates hashing, simple XOR of all rows of all the gate tables provides a secure hash. This allows simple implementation in addition to the performance improvement.

5.3 Impact on Cut-and-choose

We discuss the SFE performance improvement brought by our work on the example of the state-of-the-art approach of [LP11] and [KM15]. (Subsequent improvements to [LP11], as well as C&C, covert and other GC protocols will benefit from free GC hashing correspondingly). We review the C&C choices and parameters of [LP11,AO12,KM15] in light of [GMS08] and free hashing allowed by our work. We will show that:

1. Computing and sending additional GC hashes does not increase communication cost (computation cost is minimal due to our work), but significantly reduces cheating probability (see Table 2).
2. Keeping the cheating probability constant, we improve total C&C time by $43 - 64\%$ by sending circuit hashes instead of circuits as suggested by [GMS08] (See Table 3).

For concreteness, to achieve a cheating probability of, say, 2^{-40}, the number of garbled circuits that need to be sent is n. This incurs a communication cost, in bits, of k, where $k = n\mathsf{C}$, and C is the cost of a garbled circuit.

Sending only the hashes of the garbled circuits in the beginning of the cut-and-choose, let the total number of garbled circuits be $\tilde{n}$. Let h be the size of the hash of a GC, which is the communication cost of a check circuit. Now, we have that the communication bits incurred,

$$\tilde{k} = \tilde{n}\mathsf{h} + \frac{1}{2}\tilde{n}\mathsf{C}$$

Setting the communication complexity to be the same, $\tilde{k} = k = n\mathsf{C}$, we have,

$$n = \tilde{n}\mathsf{q} + \frac{\tilde{n}}{2}$$

where $\mathsf{q} = \frac{\mathsf{h}}{\mathsf{C}}$ is the ratio of the cost of a check circuit and the cost of a garbled circuit. For $q < 1/2$, we have $\tilde{n} = \frac{n}{q+\frac{1}{2}} > n$, thus giving a cheating probability $2^{-\tilde{n}} < 2^{-n}$ for the same communication complexity. For large circuits, we expect $\mathsf{C} \gg \mathsf{h}$, giving concrete improvements in the security at no additional communication cost.

	Communication k	Number of circuits	Cheating probability / Deterrence
[LP11]	$k = 125\|GC\|$	$n = 125$	2^{-40}
[LP11] with $\mathsf{h}\mathcal{G}$, $q = 1/4$	$k = 125\|GC\|$	$\tilde{n} = 166$	2^{-51}
[LP11] with $\mathsf{h}\mathcal{G}$, $q = 1/8$	$k = 125\|GC\|$	$\tilde{n} = 200$	2^{-62}
[KM15] without [GMS08][5]	$k = 10\|GC\|$	$n = 10$	0.9
[KM15] with $\mathsf{h}\mathcal{G}$, $q = 1/4$	$k = 10\|GC\|$	$\tilde{n} = 36$	0.972
[KM15] with $\mathsf{h}\mathcal{G}$, $q = 1/8$	$k = 10\|GC\|$	$\tilde{n} = 72$	0.986

Table 2

[5] We note that [KM15] incorporates the [GMS08] hashing in the protocol. As we discussed, sending circuits over a fast channel may only be about 3× slower than hardware-assisted garbling, while computing SHA1 may be up to 6× slower than such garbling. Hence, sending circuits over a fast channel may actually be faster than generating SHA1 hash. Therefore, in our calculations for the fast channel setting as above, we consider [KM15] without [GMS08] hash.

Performance improvement for constant cheating probability. Consider the task of evaluating a billion-gate circuit (cf. [KSS12]). We show estimated improvement due to our technique as applied to [LP11] and [KM15]. We do this in terms of expended time by unifying the computation and communication costs of generating and sending garbled circuits. These calculations are not based on specific implementations or protocol definitions. Instead they are based on simple estimates of time needed to generate, hash and send GCs, and adding them together.

We first calculate and explain the computation and communication costs in seconds of our basic tasks.

According to [BHKR13], using JustGarble to garble the AES circuit (6660 non-XOR gates) takes 637 microseconds. Adjusting for size, we calculate that the time taken for GC generation for a circuit with 1 billion gates to be 95 seconds. For communication, assuming ideal scenario in 1Gbps channel, assume we can send 1 billion bits/sec. Thus the time to send a circuit of 1 billion gates is 256 seconds at (assuming half gates and 2×128 bits per gate).

The total number of seconds needed in the cut-and-choose phase to maliciously evaluate a 1 billion-gate circuit with 2^{-40} cheating probability using previous technique and our construction using the optimal parameters. In our calculation we include the costs of generating, hashing (in our scheme) and sending the GCs. We do not include the cost of *regenerating the check circuits* at the evaluator's end that is incurred by our technique. This is because this cost is also incurred by other techniques. Indeed, checking correctness of a circuit that the evaluator already has (directly, or when using [GMS08] hash) is simplest and fastest by receiving its generating seed, reconstructing and comparing. We are concerned only with the cut-and-choose phase, and ignore the time taken for OT and GC evaluation in the protocol and show how our construction allow for reduced execution time in the cut-and-choose phase.

The cost in seconds calculated in Table 3 is obtained by adding the time to generate, hash (if needed) and send all the required garbled circuits. As explained above, we assume that it takes 95 seconds to generate a 1-Billion gate GC, and 256 seconds to send it.

Finally, we note that even though we don't know whether the dual-execution C&C of Huang et al. [HKE13] could be modified to take advantage of our Free Hash, we point out that an improved balance between the check and evaluation circuits is possible when [HKE13] is used with the [GMS08] hash. We include the calculations of optimal parameters for [HKE13] in Appendix A.

	Total number of circuits	Number of check circuits	Circuits sent	Time (in secs)
[LP11]	125	75	125	43875
[LP11] $+\mathsf{h}\mathcal{G}$	125	75	50	24675

Table 3: A billion-gate circuit. Execution time estimates of cut-and-choose with our improvements to achieve cheating probability of 2^{-40}

	Total number of circuits	Number of check circuits	Circuits sent	Time (in secs)
[AL07]	10	9	10	3510
[AL07]+$\mathsf{h}\mathcal{G}$	10	9	1	1260
[KM15] without [GMS08] [5]	10	9	10	3510
[KM15]+$\mathsf{h}\mathcal{G}$	10	9	1	1260

Table 4: A billion-gate circuit. Execution time estimates of cut-and-choose with our improvements to achieve deterrence of $\epsilon = 0.9$.

Acknowledgments. We thank the anonymous reviewers of Eurocrypt 2017 for valuable comments. We also thank Mike Rosulek for a discussion on the applicability of the Free Hash. This work was supported by the Office of Naval Research (ONR) contract number N00014-14-C-0113.

References

AL07. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, Heidelberg, February 2007.

AO12. Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 681–698. Springer, Heidelberg, December 2012.

BCC88. Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

BHKR13. Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796. ACM, 2012.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM, 1990.

BÖS11. Joppe W. Bos, Onur Özen, and Martijn Stam. Efficient hashing using the AES instruction set. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 507–522. Springer, Heidelberg, September / October 2011.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

Bra. Danny Bradbury. Bitcoin mining difficulty soars as hashing power nudges 1 petahash. `http://www.coindesk.com/bitcoin-mining-difficulty-soars-hashing-power-nudges-1-petahash/`. Retrieved Feb 3, 2017.

Bra13. Luís T. A. N. Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique - (extended abstract). In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 441–463. Springer, Heidelberg, December 2013.

BRS02. John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335. Springer, Heidelberg, August 2002.

CG13. Ran Canetti and Juan A. Garay, editors. *CRYPTO 2013, Part II*, volume 8043 of *LNCS*. Springer, Heidelberg, August 2013.

CGM16. Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 499–530. Springer, Heidelberg, August 2016.

CPS08. Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In Wagner [Wag08], pages 1–20.

FVK+15. Ben A. Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M. Bellovin. Malicious-client security in blind seer: A scalable private DBMS. In *2015 IEEE Symposium on Security and Privacy*, pages 395–410. IEEE Computer Society Press, May 2015.

GG14. Juan A. Garay and Rosario Gennaro, editors. *CRYPTO 2014, Part II*, volume 8617 of *LNCS*. Springer, Heidelberg, August 2014.

GGO+. Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Sean Gulley, and Wajdi Feghali. Improving OpenSSL performance. `https://software.intel.com/sites/default/files/open-ssl-performance-paper.pdf`. Retrieved Feb 3, 2017.

GKWY19. Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. Cryptology ePrint Archive, Report 2019/074, 2019. `https://eprint.iacr.org/2019/074`.

GMS08. Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In Smart [Sma08], pages 289–306.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

HKE13. Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In Canetti and Garay [CG13], pages 18–35.

HKK+14. Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. In Garay and Gennaro [GG14], pages 458–475.

IKNP03. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

JKO13. Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 955–966. ACM, 2013.

KKL+16. Vladimir Kolesnikov, Hugo Krawczyk, Yehuda Lindell, Alex J. Malozemoff, and Tal Rabin. Attribute-based key exchange with general policies. Cryptology ePrint Archive, Report 2016/518, 2016. `http://eprint.iacr.org/2016/518`.

KKW16. W. Sean Kennedy, Vladimir Kolesnikov, and Gordon Wilfong. Overlaying circuit clauses for secure computation. Cryptology ePrint Archive, Report 2016/685, 2016. `http://eprint.iacr.org/2016/685`.

KM15. Vladimir Kolesnikov and Alex J. Malozemoff. Public verifiability in the covert model (almost) for free. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 210–235. Springer, Heidelberg, November / December 2015.

KMR14. Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Garay and Gennaro [GG14], pages 440–457.

KMRR15. Vladimir Kolesnikov, Payman Mohassel, Ben Riva, and Mike Rosulek. Richer efficiency/security trade-offs in 2PC. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 229–259. Springer, Heidelberg, March 2015.

Kol05. Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 136–155. Springer, Heidelberg, December 2005.

KS08a. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgrard, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

KS08b. Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In Gene Tsudik, editor, *FC 2008*, volume 5143 of *LNCS*, pages 83–97. Springer, Heidelberg, January 2008.

KS16. Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 699–728. Springer, Heidelberg, May 2016.

KSS12. Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 14–14, Berkeley, CA, USA, 2012. USENIX Association.

Lin13. Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Canetti and Garay [CG13], pages 1–17.

LMS16. Helger Lipmaa, Payman Mohassel, and Saeed Sadeghian. Valiant's universal circuit: Improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017, 2016. `http://eprint.iacr.org/2016/017`.

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

LP11. Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, Heidelberg, March 2011.

LR14. Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Garay and Gennaro [GG14], pages 476–494.

Mal. Alex J. Malozemoff. libgarble: garbling library based on justgarble. `https://github.com/amaloz/libgarble`.

NPS99. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

PKV+14. Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374. IEEE Computer Society Press, May 2014.

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009.

Rab77. Michael O. Rabin. Digitalized signatures. Foundations of secure computation. In *Richard AD et al. (eds): Papers presented at a 3 day workshop held at Georgia Institute of Technology, Atlanta*, pages 155–166. Academic, New York, 1977.

RS08a. Phillip Rogaway and John P. Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In Wagner [Wag08], pages 433–450.

RS08b. Phillip Rogaway and John P. Steinberger. Security/efficiency tradeoffs for permutation-based hashing. In Smart [Sma08], pages 220–236.

Sma08. Nigel P. Smart, editor. *EUROCRYPT 2008*, volume 4965 of *LNCS*. Springer, Heidelberg, April 2008.

TJ11. Henk C. A. Tilborg and Sushil Jajodia. *Encyclopedia of Cryptography and Security*. Springer Publishing Company, Incorporated, 2nd edition, 2011.

Val76. Leslie G. Valiant. Universal circuits (preliminary report). In *STOC*, pages 196–203, New York, NY, USA, 1976. ACM Press.

Wag08. David Wagner, editor. *CRYPTO 2008*, volume 5157 of *LNCS*. Springer, Heidelberg, August 2008.

Wan18. Xiao Wang. Personal communication, 2018.

Wik. Wikipedia. Electronic signatures and law. `https://en.wikipedia.org/wiki/Electronic_signatures_and_law`. Retrieved Sept 21, 2016.

Win84. Robert S Winternitz. A secure one-way hash function built from des. In *Security and Privacy, 1984 IEEE Symposium on*, pages 88–88. IEEE, 1984.

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

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

A Performance Calculation for the Protocol of [HKE13]

The idea in the protocol of [HKE13] is to have both parties play the role of the circuit constructor and circuit evaluator respectively in two simultaneous executions of a cut-and-choose of the protocol. The protocol is thus symmetric, and symmetric protocols might be desirable in certain situations since they have less idle time. The number of garbled circuits required in the cut-and-choose to achieve statistical security $2^{-\kappa}$ is $\kappa + O(\log \kappa)$. In the cut-and-choose of [HKE13], a party successfully cheats if it generates exactly $n - c$ incorrect circuits and none of them is checked by the other party. The probability that a cheating garbler succeeds,

$$\Pr[\mathcal{A} \text{ wins}] = \frac{1}{\binom{n}{c}}$$

where n is the number of garbled circuits and c is the number of check circuits. It is easy to see that the above probability is minimized by setting $c = n/2$. This gives $\Pr[\mathcal{A} \text{ wins}] = 2^{-n+\log n}$. We now apply the hash optimization in the cut-and-choose phase of the protocol. And now, we want compute the optimal value of c in the case where the communication cost of a circuit is a check circuit is cheaper than the cost of a garbled circuit that is evaluated. Let h be the cost of the hash of a garbled circuit. The cost of a check circuit is just the hash and the cost of an evaluation circuit is the cost of the hash plus the cost of a garbled circuit. Let $\mathsf{q} = \frac{\mathsf{h}}{\mathsf{C}}$ be the ratio of the cost of a check circuit and the cost for an evaluation circuit, where $\mathsf{C} = \mathsf{h} + |GC|$. We have, the total communication complexity,

$$k = c\mathsf{h} + e\mathsf{C}$$

where e is the number of evaluation circuits, c the number of check circuits, $n = c + e$ the total number of circuits. Now, for a fixed k, given q we find the c and n that minimizes

$$\Pr[\mathcal{A} \text{ wins}] = P = \frac{1}{\binom{n}{c}}$$

Using Stirling's approximation, we get,

$$P \approx \frac{(n-c)^{n-c+\frac{1}{2}} c^{c+\frac{1}{2}}}{n^{n+\frac{1}{2}}}$$

Let $r = \frac{c}{n}$ be the optimal fraction. Using $k = c\mathsf{h} + e\mathsf{C}$ and $\mathsf{q} = \frac{\mathsf{h}}{\mathsf{C}}$, differentiating P with respect to c, and setting the first derivative to 0, we get,

$$r = (1-r)^{\mathsf{q}} \tag{9}$$

When $\mathsf{q} = 1$, this gives $r = \frac{c}{n} = \frac{1}{2}$ which is indeed the optimal value when no hashes are used and a check circuit costs the same as an evaluation circuit. Now we compare the standard cut-and-choose with the cut-and-choose using hash and using optimal parameters as computed above. For security $2^{-\kappa}$, $\kappa + O(\log \kappa)$ circuits need to be sent in the standard cut-and-choose, which gives a communication $k = |GC|(\kappa + O(\log\kappa))$, (for each party) where $|GC|$ is the cost of a garbled circuit. Now given the cost of a hashed GC to be h, we get cost of a check circuit $= \mathsf{h}$, $\mathsf{C} = |GC| + \mathsf{h}$,$\mathsf{q} = \frac{\mathsf{h}}{\mathsf{C}}$. We now solve (9) for r and set

$$n = \frac{k}{r\mathsf{h} + (1-r)\mathsf{C}} \text{ and } c = rn$$

This achieves a better cheating probability for the same communication k. In the table below, we compare the cheating probability for values of k and q. Recall, in the protocol, both parties act as sender and send $\kappa + O(\log\kappa)$ number of circuits each. The first column in Table 5 denoting the communication is the *total* communication of the cut-and-choose.

	Communication k	Optimal number of circuits	Optimal number of check circuits	Cheating probability
[HKE13]	$k \approx 90\|GC\|$	$n = 45$	$c = n/2$	2^{-40}
[HKE13] + [GMS08] hash, $q = 1/4$	$k \approx 90\|GC\|$	$\tilde{n} = 71$	$c = 0.7\tilde{n} \approx 49$	2^{-60}
[HKE13] + [GMS08] hash, $q = 1/8$	$k \approx 90\|GC\|$	$\tilde{n} = 98$	$c = 0.8\tilde{n} = 78$	2^{-68}

Table 5