The Cut-and-Choose Game and its Application to Cryptographic Protocols

Ruiyu Zhu
Indiana University

Yan Huang
Indiana University

Jonathan Katz
University of Maryland

abhi shelat
Northeastern University

Abstract

The *cut-and-choose* technique plays a fundamental role in cryptographic-protocol design, especially for secure two-party computation in the malicious model. The basic idea is that one party constructs n versions of a message in a protocol (e.g., garbled circuits); the other party randomly *checks* some of them and *uses* the rest of them in the protocol. Most existing uses of cut-and-choose fix *in advance* the number of objects to be checked and in optimizing this parameter they fail to recognize the fact that checking and evaluating may have dramatically different costs.

In this paper, we consider a refined cost model and formalize the cut-and-choose parameter selection problem as a constrained optimization problem. We analyze "cut-and-choose games" and show equilibrium strategies for the parties in these games. We then show how our methodology can be applied to improve the efficiency of three representative categories of secure-computation protocols based on cut-and-choose. We show improvements of up to an-order-of-magnitude in terms of bandwidth, and 12–258% in terms of total time. Source code of our game solvers are available to download at https://github.com/cut-n-choose.

1 Introduction

Most efficient implementations for secure two-party computation in the semi-honest setting rely on *garbled circuits*. One party, acting as circuit generator, prepares a garbled circuit for the function of interest and sends it to the other party along with garbled values corresponding to its input. The second party, who will serve as the circuit evaluator, obtains garbled values for its own inputs using oblivious transfer, and then evaluates the garbled circuit to obtain the result.

The primary challenge in handling *malicious* adversaries is to ensure that the garbled circuit sent by the first party is constructed correctly. The *cut-and-choose paradigm* is a popular and efficient mechanism for doing so. The basic idea is that the circuit generator produces and sends *several* garbled circuits; the circuit evaluator *checks* a random subset of these, and *evaluates* the rest to determine the final result. Since its formal treatment by Lindell and Pinkas [LP07], numerous works have improved various aspects of the cut-and-choose methodology and used it to design secure protocols [Woo07, LPS08, NO09, PSSW09, SS11, LP12, **?**, FJN$^+$13, Lin13, HKE13, SS13, Bra13, AMPR14, LR14, HKK$^+$14, AHMR15, **?**]. These prior works fall roughly into three categories:

1. **MajorityCut.** Here the circuit evaluator determines its output by taking the majority value among the evaluated garbled circuits. Thus, security holds as long as a majority of the evaluated circuits are correct. This is the classic approach adopted by many papers [LP07, Woo07, LPS08, LP12] and implementations [PSSW09, SS11, **?**].
2. **SingleCut.** Here the circuit evaluator is able to obtain the correct output as long as *at least one* of the evaluated circuits are correctly generated. Schemes adopting this approach include [Lin13, HKE13, Bra13, AMPR14].
3. **BatchedCut.** This considers a slightly different setting in which the parties repeatedly evaluate some function, and the goal is to obtain good amortized efficiency by batching the cut-and-choose procedure either across multiple instances of secure computation [LR14, HKK$^+$14], or at the gate level [NO09, FJN$^+$13].

Although SingleCut is asymptotically better than MajorityCut, some SingleCut protocols [Lin13] require using MajorityCut on a smaller circuit as a sub-routine, and therefore optimizations to MajorityCut can result in efficiency improvements to SingleCut. In addition, MajorityCut works better for applications with long outputs as its cost does not grow with output length.

Table 1: Bandwidth cost ratios r in various settings.

	AES[a]	Floating pt mult[b]	ORAM R/W[d]	Sort[c]
# AND gates	6800	4300	350 K	6.3×10^9
Ratio r	4533	2866	233 K	4.2×10^9

To be conservative in estimating r, figures assume 128-bit labels. [a]One-block AES128 with 128-bit wire labels. (Non-free gate counts, 6800, reported in [BHKR13, ?], hence $6800 \times 256/(256+128) \approx 4533$). [b]A single multiplication of two 64-bit IEEE-754 floating-point numbers [LWN$^+$15]. [c]Securely compute an oblivious access to an ORAM of one million 32-bit numbers [LWN$^+$15]. [d]Sorting one million 32-bit numbers [LWN$^+$15].

When setting parameters for cut-and-choose protocols, in order to optimize efficiency for some target level of security, state-of-the-art approaches treat circuit checking roughly as expensive as circuit evaluation, and hence strive to optimize the total number of garbled circuits involved. Although some researchers [Lin13, GMS08, AMPR14] observed the asymmetry in the cost of checking and evaluation, they did not explore the cost asymmetry further, and did not investigate the possibility of optimizing cut-and-choose parameters based on this asymmetry.

As evidenced by many recent prototypes [LWN$^+$15, ?, ?, ?, ?, SS11, PSSW09, ?, ?], network communication has become the most prominent bottleneck of garbled-circuit protocols, especially when exploiting dedicated hardware [BHKR13, ?] or parallelism [?, NWI$^+$15, ?] for faster garbling/evaluation. However, the bandwidth costs are markedly different for checking and evaluating circuits: garbled circuits that are evaluated must be transmitted in their entirety, but checking garbled circuits can be done by generating the circuit from a short seed and committing to the circuit using a succinct commitment [GMS08, ?, AMPR14]. Table 1 presents, in the context of a few example applications, the bandwidth costs for sending an entire circuit (i.e. the costs for an evaluated circuit) versus the cost for committing to the circuit (which, for simplicity, requires only one SHA256 hash), and thus a sample ratio r that we use in this paper as a variable.

Based on these observations, we propose a new approach to optimizing parameters in cut-and-choose protocols. Our approach casts the interaction between the circuit generator and circuit evaluator as a game, computes the optimal strategies in this game (which, interestingly, turn out to be mixed strategies), and then sets parameters while explicitly taking into account the relative costs of circuit checking and circuit evaluation. Our optimizations result in cut-and-choose approaches that can be easily integrated into prior protocols, and can reduce the bandwidth in these protocols by an order-of-magnitude in some settings.

1.1 Prior Work

The protocol of Lindell and Pinkas [LP07] checks exactly half the circuits, an idea followed in many subsequent works [PSSW09, LP12]. They showed that by generating n circuits and checking a random subset of size $n/2$, a cheating generator succeeds in convincing the evaluator to accept an incorrect output with probability at most $2^{-0.311n}$. Thus, to achieve (statistical) security level 2^{-40}, their protocol requires 128 garbled circuits. Shen and Shelat [SS11] slightly improved the bound to $2^{-0.32s}$ by opening roughly 60% (instead of one half) of the circuits; this reduces the number of garbled circuits needed to 125 for 2^{-40} security. These protocols belong to the MajorityCut category in our terminology.

The idea of using SingleCut protocols was subsequently introduced [Lin13, Bra13, AMPR14]. Here, the evaluator chooses whether to check each circuit with independent probability 1/2; now n circuits suffice to achieve security level 2^{-n}.

Most recently, several works [LR14, HKK$^+$14, ?] have proposed to amortize the cost of cut-and-choose across multiple evaluations of the same function. Along with the LEGO family of protocols [FJN$^+$13, NO09] that amortize checks at the gate level (rather than the circuit level), they all fall in the class of BatchedCut protocols. These works show that cut-and-choose can be very efficient in an amortized sense, requiring fewer than 8 circuits per execution to achieve 2^{-40} security when amortizing over 1000 executions. A brief explanation of the BatchedCut idea is given at the beginning of Section 3.3.

1.2 Contributions

We introduce a game-theoretic approach to study cut-and-choose in the context of secure-protocol design. The simplest version of cut-and-choose can be treated as a *zero-sum* game (where the utilities are 0/1 for the loser/winner) between the evaluator and the generator in which the generator wins if it can produce enough incorrect circuits to skew the protocol without being detected. Finding an optimal strategy for the evaluator can be cast as solving a linear-program and results in a randomized strategy for choosing the number of circuits to check. This linear program can be further refined to take into consideration the different cost of checking vs evaluating (i.e., the ratio r). Analyzing the equilibrium of this game leads to a constrained optimization problem that can be used to derive more efficient protocols meeting a targeted security bound (e.g. $\varepsilon = 2^{-40}$ as per many published implementations).

Our techniques enable optimization based on the precise relative costs of checking and evaluating, which in turn may depend on the function being computed as well as characteristics of specific deployment settings, such as software, hardware configuration and network condition, etc. This provides the ability to "tune" protocols to specific applications in a much more fine-grained way than before. We demonstrate that doing so can lead to bandwidth savings of 1.2–10×.

We concretely apply our methodology to three representative types of cut-and-choose-based secure-computation protocols, and show a significant overall improvement in the bandwidth usage. For example, we are able to reduce the network traffic by up to an order-of-magnitude in comparison with the state-of-the-art SingleCut (see Figure 5) and MajorityCut (see Figure 2) protocols, and savings of 20% $\sim$ 80% for state-of-the-art (already highly optimized) BatchedCut protocols (see Figure 8). Our improvements do not require any additional cryptographic assumptions and come with little development overhead.

2 Overview

Notation. Throughout this paper, we implicitly fix the semantic meaning for a few frequently-used variables (unless explicitly noted otherwise) as in Table 2.

Table 2: Frequently-used variables

ε	Failure probability of the cut-and-choose game
r	Cost ratio between circuit evaluation and checking
n	Total number of circuit copies ($n = k + e$)
k	Number of circuit copies used for checking
e	Number of circuit copies used for evaluation
b	Number of bad circuit copies generated
T	Total number of circuits used in BatchedCut.
B	Bucket size in BatchedCut.
τ	Evaluator's detection rate checking a bad gate/circuit

2.1 Problem Abstraction

Let e and k be the numbers of evaluate-circuits and check-circuits, respectively. Let r be the ratio between the costs of evaluating and checking a circuit. In the case when the parameters e,k are set deterministically and public, the cut-and-choose parameter optimization problem can be expressed as the following non-linear programming problem:

$$\arg\min_{e,k}\ r \cdot e + k$$

subject to

$$\max_b \Pr_a(e,k,b) \leq \varepsilon,$$

where ε, r are known input constants; $\Pr_a(e,k,b)$ is the probability of a successful attack; and b is the total number of bad circuits generated by the malicious generator.

In the case when at least one of the two parameters (e and k) is randomly picked by the circuit evaluator from some public distributions (but sampled values remain secret to the circuit generator at the time of circuit generation), the optimization problem takes a more general form

$$\arg\min_{S_E}\ \mathbb{E}[cost(r, S_E)]$$

subject to

$$\mathbb{E}\left[\Pr_a(S_E, S_G)\right] \leq \varepsilon, \quad \forall\, S_G$$

where S_E and S_G are the circuit evaluator's and the circuit generator's strategies, respectively; *cost* is the cut-and-choose cost function, and $\mathbb{E}[\cdot]$ denotes the expectation function. Note that the cost function does not need to account for pre-maturely terminated protocol executions (due to detected cheating activity). Our goal is to identify the best S_E for the evaluator. We leave the notion of S_E and S_G abstract for now but will give more concrete representations when analyzing specific protocols in Section 3.

We stress that, in contrast to the common belief used in the state-of-the-art cost analysis of cut-and-choose protocols, the cost of cut-and-choose is usually not best represented by n—the total number of circuits generated, but rather by a cost ratio r between checking and evaluation which depends on many factors such as (1) the kind of cost (e.g., bandwidth or computation); (2) the deployment environment (e.g., network condition, distribution of computation power on the players, buffering, etc.) (3) the specific cryptographic primitives and optimization techniques (e.g., the garbling scheme) used in a protocol. Therefore, the best practice would be always micro-benchmarking the ratio between the per circuit cost of evaluation and checking before running the protocol, and then select the best cut-and-choose strategies accordingly.

2.2 Summary of Our Results

The main thesis of this work is,

> *Cut-and-choose protocols should be appropriately configured based on the security requirement (ε) and the cost ratio (r) benchmarked at run-time. Such practice can bring significant cost savings to many cut-and-choose based cryptographic protocols.*

To support our thesis, we have formalized the cut-and-choose-based protocol configuration problem into a constrained optimization problem over a refined cost model. Our solutions to the constrained optimization problem imply randomized strategies are optimal. We show how to support randomized strategies in the state-of-the-art cut-and-choose-based cryptographic protocols with only small changes. We applied this methodology to analyze three major types of cut-and-choose schemes and the experimental results corroborate our thesis. We have implemented a search tool for each category of schemes to output the optimal parameters. The tool is available at https://github.com/cut-n-choose.

3 Case Studies

In this section, we show how our general idea can be applied to three main types of two-party secure computation protocols that are based on the cut-and-choose method to substantially improve their performance. We assume that n is fixed and public, while e will be selected from some distribution and remain hidden to the generator until all circuits are generated and committed.

3.1 MajorityCut Protocols

MajorityCut strategy stems from an intuitive folklore idea: the circuit evaluator randomly selects k (out of a total n circuits) to check for correctness, evaluates the remaining $e = n - k$ circuits, and outputs the *majority* of the e evaluation results. All previous work assumed the use of fixed and public n, e, k parameter values, which grants a malicious generator unnecessary advantages. For example, knowing e, a malicious generator can choose to generate $\lceil e/2 \rceil$ bad circuits to maximize the chance that an honest evaluator outputs a wrong result. Thanks to its simplicity, it is the scheme the most widely adopted by implementations thus far.

In the following, we show how to apply our observations to MajorityCut protocols, which involves delaying the revelation of cut-and-choose parameters and employing a mixed strategy (instead of a pure one) to minimize the total cost of cut-and-choose.

Analysis. We represent the evaluator's strategy by a vector $\vec{x} = (x_0, x_1, \ldots, x_n)$ where x_i is the probability that the evaluator evaluates i uniform-randomly chosen circuits and checks the remaining $n - i$. The expected cost of MajorityCut is

$$\sum_{i=0}^{n} \left[x_i \cdot (i \cdot r + (n-i)) \right] = n + (r-1) \sum_{i=0}^{n} x_i \cdot i$$

If the generator produces b incorrect circuits and the evaluator evaluates i circuits, the probability that the evaluator's check passes is $\binom{n-b}{n-i} \big/ \binom{n}{n-i}$. After a successful check, the evaluator loses the security game if and only if $2b \geq i$, i.e., there is no majority of correct evaluation circuits. Hence, when the evaluator uses strategy $\vec{x}$, the expected failure probability of the MajorityCut scheme is

$$\sum_{i \leq 2b} x_i \cdot \binom{n-b}{n-i} \Big/ \binom{n}{n-i}$$

Since $i \leq n$ and $\binom{n-b}{n-i} = 0$ for all $i < b$, this sum can be further reduced to $\sum_{i=b}^{\min(n,2b)} x_i \cdot \binom{n-b}{n-i} \big/ \binom{n}{n-i}$. The security requirement stipulates that for every choice of b by the malicious generator, the resulting cut-and-choose failure probability should be less than ε. In other words, the goal of picking optimal cut-and-choose parameters can be achieved by solving the following linear program:

$$\min_{\vec{x}} \; n + (r-1) \sum_{i=0}^{n} x_i \cdot i$$

subject to

$$\begin{aligned} x_i &\geq 0 \\ \sum_{i=0}^{n} x_i &= 1, \\ \sum_{i=b}^{\min(n,2b)} x_i \cdot \binom{n-b}{i-b} \Big/ \binom{n}{i} &< \varepsilon, \; \forall b \in \{1, \ldots, n\}. \end{aligned}$$

Solving this linear program provides us an equilibrium strategy for every fixed n, ε, r. Using standard LP solvers, such programs can be solved exactly for n that ranges into the thousands (i.e., all practical settings).

With this capability, we can identify, for a given target ε and ratio r, the optimal n (that leads to the least overall cost) by solving the linear programs for all feasible n values. While this leads to the search algorithm described in Figure 1, we note several important observations that speedup the search:

1. We begin our search at $n_0 = \lfloor \varepsilon \rfloor$ and consider $n = n_0, n_0 + 1, \ldots$. After solving each LP, we identify a current best cost c^*. Observe that $c^* - (r-1)$ is an upper-bound of the best n (noted n^*), since any feasible strategy with $n > c^* - (r-1)$ will cost at least c^* (the evaluator need to evaluate at least one circuit except with at most ε probability). Thus, as our search continues, we update c^*, and terminate the search as soon as all values of n between n_0 and $c^* - (r-1)$ are examined.

2. When the value of r is beyond moderate (i.e., $r > t_r$ for some constant t_r like 128 with our laptop), searching for the optimal cost becomes time-consuming as it involves solving the above linear programming problem for many relatively large n values (e.g., $n > 500$). In these settings, however, we opt to live with a sub-optimal *pure* strategy, based on the observation that the standard deviation of e is already so small (less than 0.6 and only keeps decreasing as r grows) that the cost of a sub-optimal pure strategy (i.e. a combination of n and e) approximates the theoretical optimal pretty well (Figure 3b).
3. We note that when $r > t_r$ (step 2), it suffices to search all e less than e_0 (recall (e_0, n_0, c_0) is the starting point of our search, simply derived from the traditional setup with MajorityCut) instead of the infinite range because any strategy with $e = e_0 + 1$ that is more efficient than one with $e = e_0$ has to use at least $r - 1$ fewer check circuits. Since $t_r = 128$, such strategy would have used at least 127 fewer check-circuits, which will contradict with $n_0 = 128$ (assuming $\varepsilon = 2^{-40}$).

Results. We have implemented the search algorithm of Figure 1 and run it with a wide range of practically possible r values (see Figure 2). For r values ranging from 5000 to 10^9, which are typical regarding the cost in network traffic, we can achieve 6 to 16 times savings compared to traditional MajorityCut protocols. Even when r is small, such as $8 \sim 128$ which are representative when considering only the timing cost, our approach brings about 1.45 to 3 times savings. When circuit-level parallelism is exploited like in the work of [?, ?, ?], where r typically ranges from 50 to 500 (see Table 5), we are able to speedup the best existing works by $2.3 \sim 3.9$ times.

Table 3 gives two example optimal strategies for achieving $\varepsilon = 2^{-40}$ security when $r = 10$ and $r = 100$, respectively. We observe that the solution mixes fewer pure strategies (which is consistent to the decrease of variance) as r grows. Also note that all pure strategies with even es are dominated by ones with odd es. In contrast to current implementations, these strategies suggest that the generator produce a few hundred circuits, but only send roughly 13–21 of them. (Such a scheme is for example quite feasible when using the GPU to produce and commit to garbled circuits.) In comparison, the best protocols that use BatchedCut need to amortize 1000s of protocol executions to achieve security when sending roughly 10 circuits.

$r = 10$			$r = 100$		
n	i	x_i as %	n	i	x_i as %
361	7	$1 \cdot 10^{-4}$	514	3	$4 \cdot 10^{-6}$
	9	$9 \cdot 10^{-4}$		5	$2.04 \cdot 10^{-4}$
	11	$7 \cdot 10^{-3}$		7	$7.44 \cdot 10^{-3}$
	13	$4.54 \cdot 10^{-2}$		9	0.21
	15	0.25		11	4.86
	17	1.23		13	94.73
	19	5.36		15	0.19
	21	20.9			
	23	72.2			
Saves 13.5% b/w			Saves 65.3% b/w		

Table 3: Example optimal strategies for MajorityCut protocols. ($\varepsilon = 2^{-40}$, only non-zero x_is are listed)

In Figure 2, the cross-marked solid curve delineates the optimal cost of mixed strategies (among all strategies with public fixed n), while the dot-marked dashed curve delineates pure-strategy approximation of the optimal mixed strategies (efficiently computed as a result of step 2 of Figure 1 search algorithm). We observe that the pure-strategy approximation actually improves as r get bigger. But when r is relatively small ($100 \geq r \geq 1$), our optimization-based approach can indeed bring about $1\% \sim 11\%$ extra improvement (Figure 2). Last, the curves for two different ε values have similar shape but the improvement as a result of our approach is significantly larger for smaller ε values.

We also observe from Figure 2 that the performance boost seems to be upper-bounded by some value related to ε, no matter how big r becomes. This makes some intuitive sense because the cost of our optimized protocols will be upper-bounded by a linear function of r (as the solution comes out of solving the linear programming problem where r constitutes the coefficients of the unknowns). We leave the formal proof of this intuition as an interesting future work.

To examine the characteristic of our solution more closely for $1 \leq r \leq 128$, we have plotted the comparison of overall cost of the optimal strategy with respect to best prior works (Figure 3a), the standard deviation of the overall cost (Figure 3b, recall the optimal strategy is a randomized strategy), and the best n used in every optimal strategy (Figure 3c). Note that the standard deviation of the overall cost is also exactly the standard deviation of e because the randomness in cost all comes from the randomness in selecting e. The fact that the standard deviation quickly drops to less than 0.3 (when $r \geq 100$) and strictly decreases justifies the accuracy of pure strategy approximation for large r.

Changes to Existing Protocols. Our approach to MajorityCut applies to many published cut-and-choose based

```
Input: $\varepsilon, r$.
Output: $c^*, n^*, \boldsymbol{x}^*$.
1. If $r \leq t_r$,
   (a) Initialize $n_0 := \lfloor \log \varepsilon \rfloor$ and $c^* := +\infty$.
   (b) For $n := n_0$ to $c^* - (r-1)(1-\varepsilon)$,
       i. Solve the MajorityCut linear programming problem for $(n, \varepsilon, r)$ to obtain $(c, \boldsymbol{x})$ where $c$ is the minimal cost of
          LP$(n, \varepsilon, r)$ and $\boldsymbol{x}$ represents the corresponding strategy to achieve $c$.
       ii. If $(c, \boldsymbol{x})$ is a feasible solution and $c^* > c$, then $(c^*, \boldsymbol{x}^*, n^*) := (c, \boldsymbol{x}, n)$.
   (c) Output $(c, n^*, \boldsymbol{x}^*)$.
2. If $r > t_r$,
   (a) Initialize $n_0 := 3\lfloor \log \varepsilon \rfloor, e_0 = n_0/2, c_0 := 3\lfloor \log \varepsilon \rfloor + (r-1)e_0$ and $c^* = c_0, n^* = n_0$.
   (b) For $i := 1$ to $+\infty$ until $e_{i-1} = 1$,
       i. Set $e_i = e_{i-1} - 1$ and compute the smallest $(c_i, n_i)$ that satisfies the security constraints.
       ii. If $c^* > c_i$, then $(c^*, n^*, e^*) := (c_i, n_i, e_i)$.
   (c) Output$(c, n^*, \{x_0, \ldots, x_n\})$ where $x_i = 1$ if $i = e^*$, and $x_i = 0$ otherwise.
```

Figure 1: Search the most cost-efficient strategy $(n, \boldsymbol{x})$ for MajorityCut protocols. $\log(\cdot)$ is base-2. c is the minimal cost, n is the fixed total number of circuits and $\boldsymbol{x} = (x_0, \ldots, x_n)$ stands for the evaluator's best strategy to sample e. While the value of t_r depends on the hardware and users' tolerance of performance, $t_r = 128$ works well for our MacBook Air for $\varepsilon = 2^{-40}$.

two-party computation protocols; in particular, it applies directly to those protocols in which the generator first commits to n garbled circuits, and later, after a coin-tossing protocol between generator and evaluator, opens each check circuit by sending either (a) both the 0 and 1 labels for all of its input wires, or (b) the random coins used to construct the circuit. Goyal, Smith and Mohassel [GMS08] were the first to use this technique and the second ("$I + 2C$") protocol from Kreuter, Shelat, Shen [?] also operates in this way. In these cases, no modifications to the security analysis are needed. For every specific ε and r, our solver outputs a particular n and a distribution $\vec{x}$ for picking e. Roughly speaking, the only changes needed in the protocol are straightforward: the evaluator announces this n beforehand and the result of the coin-tossing protocol ρ is used to sample e according to $\vec{x}$ using standard methods (instead of the 1/2 or 3/5 fractions as before). The simulation of a malicious evaluator proceeds as in the original security proof with the exception that the simulator first samples e according to $\vec{x}$ using random tape ρ and then (as before), uses a simulated coin-tossing to ensure the outcome of the toss induces ρ. (The coin-tossing method is a simple and effective method to prove security; other proofs may also exist.)

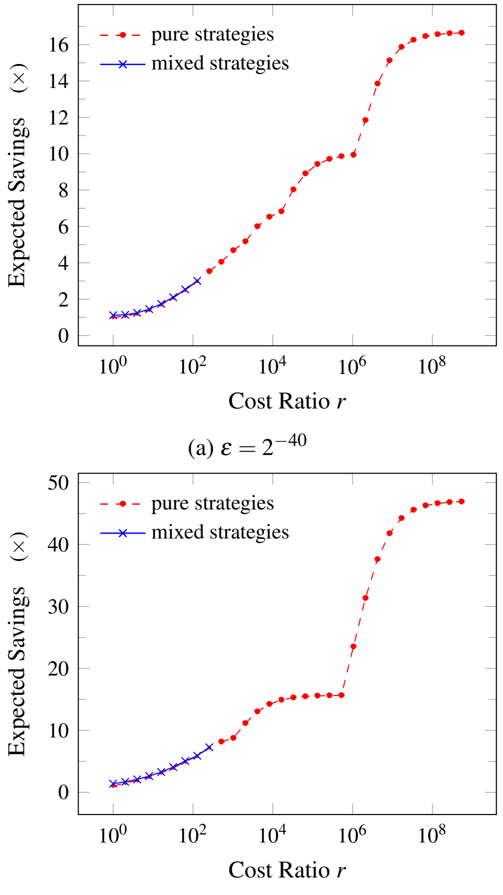
Similarly, the first protocol of Lindell and Pinkas [LP07] can be modified to adopt this idea: step (3) should send commitments to garbled circuits, modify step (4) to use the random tape from coin-tossing to sample e, modify step (8) so that the garbler sends the entire garbled circuits for the evaluation specimens as well as openings to the commitments so that the evaluator can check consistency.

The idea seems applicable to many protocols which have the property that the set of checked circuits becomes publicly verifiable. For example, Mohassel and Riva [MR13] use a different idea in their protocol to allow the same output labels to be used across all n copies of the garbled circuit. In their original protocol, the evaluator and generator then use coin-tossing to select the open circuits, but then proceed to evaluate the remaining circuits first, perform some checks, and then the evaluator commits to the output labels. Finally, the generator opens the check circuits for the evaluator to check, and if all succeed, the evaluator opens a commitment to the output. Although a different order, the modifications noted above seem to apply without the need to modify the security proof.

The protocols of Lindell and Pinkas [LP12] and shelat and Shen [SS13], however, seem to require more subtle modifications and new security arguments to use our technique. In both cases, the protocols use a special oblivious transfer (instead of coin-tossing) to allow the evaluator to independently choose the set of check circuits. In the case of [LP12], the fact that the size of the set of checked circuits is *fixed*, and therefore verifiable by the garbler, is needed in the security proof. This restriction can be lifted with a variant of cut-and-choose oblivious transfer proposed and used in Lindell's SingleCut protocol [Lin13]. For a different reason, a new security argument will also be needed for shelat and Shen [SS13].

3.2 SingleCut Protocols

With SingleCut protocols, extra cryptographic mechanisms (e.g., a second-stage fully secure computation as in [Lin13] or an additive homomorphic commitment as in [AMPR14]) are employed in order to *weaken* the



(a) $\varepsilon = 2^{-40}$

(b) $\varepsilon = 2^{-20}$

Figure 2: Our savings for MajorityCut protocols

soundness property required by the cut-and-choose technique. In particular, in such protocols, it suffices to ensure that the cut-and-choose game fails *only if all evaluation circuits selected by the evaluator are corrupted*. For example, if even one evaluation circuit is properly formed, then the evaluator will either receive the same output from all of the evaluated circuits (in which case it can accept the output since one circuit is good), or it receives two different outputs. In the latter case, the evaluator uses the two different authenticated output labels as input to an auxilliary, MajorityCut-based secure computation in order to recover the garbler's input, and then evaluate the function itself.

The state-of-the-art SingleCut protocols implicitly assume $r = 1$, in which case an honest evaluator's best strategy is to *evaluate each garbled circuit with probability 1/2*, as there is only a single way for the malicious generator to win the cut-and-choose game. In reality, however, r is not necessarily equal to 1. In order to achieve ε statistical security, this strategy will lead to an expected $\lfloor \log \varepsilon \rfloor \cdot (r+1)/2$ units of cost.

Analysis. As before, let i be the number of evaluation circuits and $\vec{x}$ be a distribution such that x_i is the probability that the evaluator chooses to evaluate i circuits. Then the cost of the cut-and-choose scheme is $n + (r-1)\sum_{i=0}^{n} x_i \cdot i$.

Fix b, the number of incorrect circuits chosen by the generator. The first observation is that when the evaluator picks $e \neq b$, then the generator certainly loses the game. When $e = b$, recall that there are $\binom{n}{b}$ different ways to select b evaluation circuits (out of n circuits in total). Assuming the evaluator uniform-randomly picks one of the $\binom{n}{b}$ ways, then the generator looses the cut-and-choose game with probability $1\big/\binom{n}{b}$ because it happens only if the generator guesses all n of the evaluator's check-or-evaluate decisions correctly. Since the event that the evaluator picks $e = b$ is independent of the event that the generator guessed all decisions correctly, the overall failure probability is $x_b\big/\binom{n}{b}$. As a result, the security requirement can be dramatically simplified in comparison to MajorityCut. In particular, we need that every pure strategy for the generator, i.e., every choice of b, wins with probability at most ε: $x_b/\binom{n}{b} < \varepsilon$.

Therefore, fixing n, r and ε, the original cut-and-choose game configuration problem can be translated into the following linear programming problem:

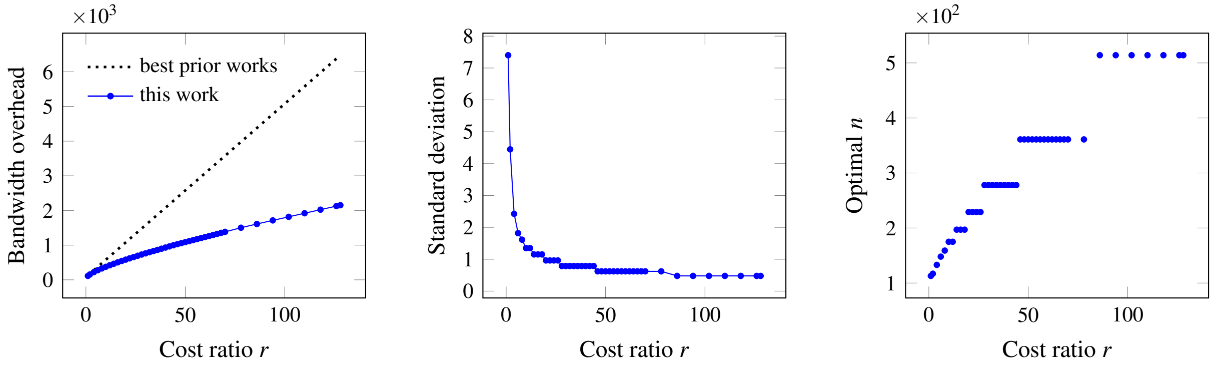
$$\min_{\vec{x}} \; n + (r-1)\sum_{i=0}^{n} x_i \cdot i$$

subject to

$$\begin{aligned} x_i &\geq 0, \quad \forall i \in \{0,\ldots,n\} \\ \sum_{i=0}^{n} x_i &= 1, \\ x_b \Big/ \binom{n}{b} &< \varepsilon, \quad \forall b \in \{0,\ldots,n\}. \end{aligned}$$

Next, we show that the linear programming problem above can actually be solved highly efficiently thanks to its special form. The key observation is that this linear programming problem is in essence a special *continuous knapsack program* (where the weight $w_i = i$). In order to minimize $\sum_{i=0}^{n} x_i \cdot i$, we aim to maximize x_i (which is upper-bounded by $\varepsilon \cdot \binom{n}{i}$ and collectively constrained by $\sum_{i=0}^{n} x_i = 1$) for all small i's. This leads to the following simple greedy algorithm that solves the problem in linear time (of n).

1. For $i = 0$ to n,
   (a) Set $x_i := \varepsilon \cdot \binom{n}{i}$.
   (b) If $\sum_{j=0}^{i} x_j \geq 1$ then set $x_i := 1 - \sum_{j=0}^{i-1} x_j$, $x_j := 0$ for all $j > i$, and return $\{x_i | 0 \leq i \leq n\}$.
2. If $\sum_{j=0}^{i} x_j < 1$, return $\perp$ (i.e., the problem has no feasible solution); otherwise, return $\{x_i | 0 \leq i \leq n\}$.



(a) (b) (c)

Figure 3: Characteristics of optimal mixed-strategy solutions for MajorityCut protocols ($\varepsilon = 2^{-40}$. The bandwidth overhead is measured in *units*. A unit cost is that of evaluating a evaluation-circuit. The standard deviation chart applies to both the overall cost and e.)

```
Input: $\varepsilon, r$.
Output: $c^*, n^*, \boldsymbol{x}^*$
1. Initialize $n_0 := \lfloor \log \varepsilon \rfloor$ and $c^* := n_0 \cdot (r+1)/2$.
2. For $n := n_0$ to $c^* - (r-1)(1-\varepsilon)$,
    (a) Solve the SingleCut linear programming problem for $(n, \varepsilon, r)$ to obtain $(c, \boldsymbol{x})$ where $c$ is the
        minimal cost of LP$(n, \varepsilon, r)$ and $\boldsymbol{x}$ represents the corresponding strategy to achieve $c$.
    (b) If $(c, \boldsymbol{x}) \neq \perp$ and $c^* > c$, then $(c^*, \boldsymbol{x}^*, n^*) := (c, \boldsymbol{x}, n)$.
3. Output $(c, n^*, \boldsymbol{x}^*)$.
```

Figure 4: Search the optimal strategy $(n, \boldsymbol{x})$ for SingleCut protocols. $\log(\cdot)$ is base-2. c is the minimal cost, n is the fixed total number of circuits and $\boldsymbol{x} = (x_0, \ldots, x_n)$ stands for the evaluator's best strategy to sample e.

As with the MajorityCut setting, we scan all possible values of n to identify the best n leading to the smallest overall cost (Figure 4). Fortunately, thanks to the high efficiency of the above linear programming solver, we are always able to identify the optimal n within seconds for r as large as 10^{10}.

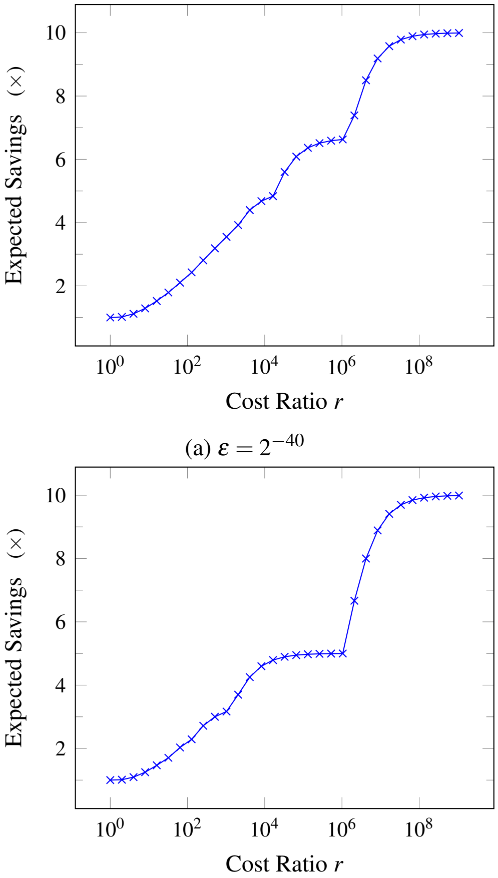
Results. Using the search algorithm described above, we are able to compute the fixed-n, variating e optimal randomized strategies for every ε and r. We summarize the performance gains in Figure 5a. The savings due to our approach rise steadily for $r < 10^4$ and can get to about 10X for reasonably large r (e.g., $r = 7 \times 10^7$, which roughly corresponds to the bandwidth-based cost-ratio for privately computing the edit distance between two 1000-character strings). Generally, it appears that the improvement-curves (Figure 5) for different ε share some similarity in their shape but smaller ε results in bigger improvements.

Table 4 shows two example optimal strategies of SingleCut protocols for $r = 10$ and $r = 100$, respectively. We observe that the optimal strategy exhibit some pattern: the number of evaluation circuits with positive support falls within $[0, g]$ where $g < n$ and g shrinks as r grows. An interesting note is that $e = 0$ (i.e., checking all n circuits) has positive support, albeit with probability less than 2^{-40}. The same issue arises in [Lin13]; in practice, one can use a special check to avoid this extremely unlikely case.

$r = 10$			$r = 100$		
n	i	x_i as %	n	i	x_i as %
65	0	$9 \cdot 10^{-11}$	180	0	$9 \cdot 10^{-11}$
	1	$5.91 \cdot 10^{-9}$		1	$1.64 \cdot 10^{-8}$
	2	$1.89 \cdot 10^{-7}$		2	$1.47 \cdot 10^{-6}$
	3	$3.97 \cdot 10^{-6}$		3	$8.69 \cdot 10^{-5}$
	4	$6.16 \cdot 10^{-5}$		4	$3.85 \cdot 10^{-3}$
	5	$7.51 \cdot 10^{-4}$		5	0.14
	6	$7.51 \cdot 10^{-3}$		6	3.95
	7	$6.33 \cdot 10^{-2}$		7	95.91
	8	0.46			
	9	2.91			
	10	16.28			
	11	80.28			
Saves 26.4% b/w			Saves 57.0% b/w		

Table 4: Example optimal strategies for SingleCut protocols. ($\varepsilon = 2^{-40}$, only non-zero x_is are listed)

Figure 6 presents a closer look at various character-



(a) $\varepsilon = 2^{-40}$

(b) $\varepsilon = 2^{-20}$

Figure 5: Bandwidth savings for SingleCut protocols

istics of the optimal strategies, including costs in units (Figure 6a), the standard deviation of the costs (which also applies to e, Figure 6b), and the ns associated with those optimal strategies (Figure 6c). We note in Figure 6b that the standard deviation for SingleCut optimal strategies are generally smaller (about half) than that for MajorityCut strategies (Figure 3b). In addition, the ns for the optimal strategies also exhibit a staircase effect like in MajorityCut. This is because for any fixed ε, it does not make sense to trade in a larger n for a smaller e, unless r exceeds certain discrete threshold values.

Changes to Existing Protocols. The changes needed in protocol 3.2 of Lindell (CRYPTO, 2013) to support our technique are standard:

- Add a step 0 to [Lin13, Protocol 2], where an n is fixed in advance based on ε and r.
- Change step 2(b) of [Lin13, Protocol 2] to: P_2 picks the check-set J at random so that $|J| = k$, where k is randomly sampled from the distribution computable (from n, r, ε) by the 2-step algorithm given above.

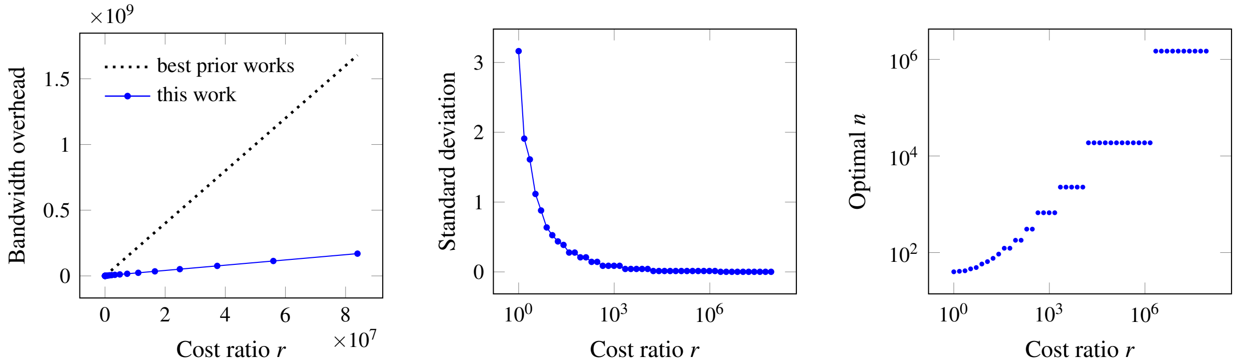
Afshar et al. [AMPR14] present a conceptually simple and elegant non-interactive secure computation protocol; it uses a cut-and-choose technique and achieves security 2^{-40} by sending 40 garbled circuits. Surprisingly, this can be done in just one round. Like Lindell, they create a trapdoor which allows an evaluator to recover the garbler's inputs with high probability if the garbler sends at least one honest circuit and one corrupted one. However, since their protocol is only 1 round, the cut-and-choose is implemented through oblivious transfer; specifically, the evaluator recovers a seed for all of the check circuits through OT, and the garbler sends all circuits in its one message. In order to apply our technique, we need to add extra rounds (in order to save substantial communication costs). Instead of sending the full circuits in the first message, we change the protocol to send succinct commitments of the circuits (thereby committing the sender) which keeps the first message short. In the next message, the evaluator asks the garbler to send the evaluation circuits only; and the evaluator uses its previous messages to continue running the original protocol. We believe these modifications are consistent with the security proof implicitly given in [AMPR14]. As a result, we can run this protocol with significantly reduced communication costs for $r > 1$ (see Figure 5).

3.3 BatchedCut Protocols

The basic idea of BatchedCut is to amortize the cost of cut-and-choose across either many protocol executions (of the same circuit) [HKK+14, LR14] or many basic gates [**?**, FJN+13, NO09] of a big circuit. Without loss of generality, we focus on the setting of batched execution of a single functionality. Roughly speaking, the evaluator randomly selects and checks k out of T circuits in total and randomly groups the remaining circuits in buckets of size B. The state-of-the-art can ensure correctness as long as at least one good circuit is included in every bucket. This can effectively reduce the number of circuit copies to less than 8 (c.f. the optimal 40 without amortization [AMPR14, Lin13]) per execution to ensure 2^{-40} security. However, optimality of this result holds only if $r = 1$. In this section, we present our approach to optimize BatchedCut protocols for general r values.

We note one technical complication in this setting: in the checking stage, a bad circuit (or gate) might only be detected by the evaluator with probability τ. Although $\tau = 1$ for most protocols, it can be less than 1 for some other protocols, e.g., $\tau = 1/2$ for [**?**] and $\tau = 1/4$ for MiniLEGO [FJN+13]. Our analysis below is generalized to account for any $0 \le \tau \le 1$.

Analysis. Let N be the number of times a particular functionality will be executed, T be the total number of circuits generated to realize the N executions, and let B denote the bucket size.



(a) (b) (c)

Figure 6: Characteristics of optimal mixed-strategy solutions for SingleCut protocols ($\varepsilon = 2^{-40}$. The bandwidth overhead is measured in *units*. A unit cost is that of evaluating a evaluation-circuit. The standard deviation chart applies to both the overall cost and e.)

With any positive r, we want to identify parameters (T, B) such that $cost(T, B)$ is minimized over all (T, B) configurations that satisfy the security constraint. That is, $\Pr_{fail}$, the overall failure probability of cut-and-choose, should be no more than ε. Therefore, the problem reduces to the following constrained optimization problem:

$$\min\ T + (r-1)BN$$

subject to

$$\Pr_{fail}(N, T, B, \tau, b) \quad < \quad \varepsilon, \quad \forall b \in \{0, \ldots, T\}$$

where b is the number of bad circuits a malicious generator chooses to inject.

Note that $\Pr_{fail}$ describes the failure across all N executions as follows: In the first move of the game, the evaluator picks $T - BN$ circuits to open and verifies all are correct. In the second move, the evaluator randomly partitions the BN unopened circuits into buckets of B circuits. A failure occurs if the adversary is able to corrupt b circuits such that the first check passes, and there is some bucket containing only corrupted circuits. We let $\Pr_c(N, B, T, \tau, b, i)$ denote the probability of the first case and $\Pr_e(N, B, b)$ denote the second.

First, as before, when i circuits are opened in the first phase, the garbler succeeds with probability

$$\Pr_c(N, B, T, \tau, b, i) = (1-\tau)^i \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}}$$

Here the extra $(1-\tau)$ term reflects the case when checking a circuit can succeed with some chance even if it is corrupt. There are T total circuits, and $T - BN - i$ of them can be checked. The next term, $\Pr_e$ reflects the probability that conditioned on the first phase passing, the evaluator randomly assigns the remaining circuits to buckets, and one bucket of size B contains all corrupted circuits. See [LR14] for a detailed discussion of this term.

$$\Pr_e(N, B, b) = \binom{b}{B} \Big/ \binom{BN}{B} \tag{1}$$

$$+ \sum_{i=0}^{b-1} \Pr_e(N-1, B, b-i) \cdot \frac{\binom{b}{B-i}\binom{BN-b}{i}}{\binom{BN}{B}} \tag{2}$$

$$\Pr_e(N, B, b) = 0, \qquad \forall 0 \le b \le B \tag{3}$$

The last equation follows because a garbler who corrupts fewer than B circuits never succeeds. Finally, since phase 1 and phase 2 are independent, we conclude that

$$\Pr_{fail}(N, B, T, \tau, b) = \sum_{i=0}^{b} \Pr_c(N, B, T, \tau, b, i) \Pr_e(N, B, b-i)$$

The summation over i occurs because every check only succeeds with probability τ, and thus even after i checks on corrupted circuits, $b - i$ corrupted circuits may remain in the second phase.

Having explained the constraint, Figure 7 describes our search algorithm to solve the BatchedCut parameter optimization problem. The basic idea is simple—for every $B = 2, 3, \ldots$, find the least T such that the security constraint is satisfied for every $b \in \{0, \ldots, T\}$. Our main contribution here is to make the search efficient enough for realistic r, N, and ε values, which is achieved based on a new efficient and accurate way to calculate $\Pr_{fail}$ (which will be detailed later in this section) and the following important observations to ensure efficiency and completeness of the search:

1. For every B, the cost $cost(T, B)$ strictly increases with T while the failure rate $\Pr_{fail}$ strictly decreases with T.

So the best T for a given B can be identified efficiently through iterative search, using both *exponential backoff* and *binary search*.

2. The constraint that $\Pr_{fail}(\cdots) < \varepsilon$ regardless of the attacker's strategy can be verified by computing $\Pr_{fail}(\cdots)$ for every $b \in \{1, \ldots, T\}$ (where b is the number of incorrect circuits generated by the attacker), which, if naively implemented, would require computing $\Pr_{fail}$ $T \cdot T$ times for every B. We can leverage the idea of generating functions to reduce it to $T+T$ inexpensive operations (we will explain this in detail in a bit).
3. Assuming $c = (T-BN)/T > c_0$ (where c_0 is a small positive constant determined solely by the evaluator), it does not make sense for a malicious generator to insert more than $b^u = -(s+1)/\log(c_0/2 + \frac{2}{2/(1-c_0)-i_0/N})$ bad circuits. We shall prove this observation as Claim 2. This observation further reduces $T+T$ down to $b^u + b^u$ inexpensive operations.
4. A smaller feasible T we found along the way stipulates an upper-bound on the B's that we need to examine, under the assumption that $(T-BN)/T > c_0$.

Compute $\Pr_{fail}$ Efficiently. For every N, B, T, τ, b, the probability of a malicious generator's successful attack can be described by equations described above. However, for most N, T values (e.g., $N > 2^{15}$), it is infeasible to compute $\Pr_c$ (which involves calculating large binomial coefficients) and $\Pr_e$ (which involves exponential number of slow recursions) based on (3.3) and (2) with accuracy comparable to ε using the equations (3.3) and (2).

Hence, we propose an efficient way to compute $\Pr_e$ and $\Pr_c$ with provable accuracy.

1. **Compute** $\Pr_e(N, B, b)$**.** The idea is to use *generating functions* to efficiently calculate $\Pr_e$ as the ratio between the number of ways to group garbled circuits into buckets that will result a failure (i.e., at least one bucket is filled with all B bad circuits) and the total number of ways to group the garbled circuits. First, we can use function $g(x,y) = (1+x)^B + (y-1)x^B$ to model the circuit assignment process for a single bucket, where 'x' denotes a "bad" gate and '1' denotes a "good" gate, thus the coefficient of x^i in $g(x,y)$ equals to the number of ways to assign i bad gates to a bucket. Note that we explicitly introduce the symbol 'y' into the coefficient of x^B to denote the event that "all B gates in a bucket are bad". Next, we introduce another generating function $G(x,y) = g(x,y)^N$ to model the circuit assignment process over all of the N buckets: the coefficient (which is a polynomial in y, hence written as $f_i(y)$) of x^i in $G(x,y)$ denotes the number of assignments that involve i bad gates in total. Let $f_i(y) = \sum_{j=0}^{\infty} c_j y^j$ (where c_j are constants efficiently computable from $G(x,y)$), then $f_b(1) = \sum_{j=0}^{\infty} c_j$ is the total number of assignments with b bad gates used in the evaluation stage; and $f_b(1) - f_b(0) = \sum_{j=1}^{\infty} c_j$ is the number of assignments (among all with b bad circuits) that result in at least one broken bucket. Hence, we compute $\Pr_e(N, B, b) = (f_b(1) - f_b(0))/f_b(1)$. We can further reduce the cost of computing the coefficients of $G(x,y)$ dramatically, by not distinguishing any terms y^{j_1} and y^{j_2} for any $j_1, j_2 \geq 1$. That is, multiplying $(u+vy)$ and $(w+ty)$ yields $uw + (ut+vw+vt)y$, hence, however big N and B are, all c_js are linear formulas of y.
2. **Compute** $\Pr_c(N, B, T, \tau, b, i)$**.** Recall that typically T, N are large while b, i are far smaller than N. So the dominating cost in computing $\Pr_c$ is to calculate $\binom{T-b}{T-BN-i} \Big/ \binom{T}{T-BN}$. To this end, we approximate $\Pr_c(N, B, T, \tau, b, i)$ using $\left(\frac{T-BN}{T}\right)^i \left(\frac{BN}{T-i}\right)^{b-i}$, whose high accuracy is formally proved in Claim 1.

To illustrate the precision of the above calculation, e.g., when $s = 40$, if $N = 50,000$, the overall error in our calculation of $\log \Pr_{fail}(N, B, T, \tau, b)$ is less than 1. Note that the error only decreases as N grows (following Claim 1 and Claim 2).

Claim 1 *Let T, B, N, b, i be defined as above. Then*

$$\lim_{N->\infty} \frac{\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} = \left(\frac{T-BN}{T}\right)^i \left(\frac{BN}{T-i}\right)^{b-i}.$$

Claim 2 *Let $\Pr_{fail}(N, B, T, \tau, b)$ be the probability that the cut-and-choose game fails in a BatchedCut scheme, (with b bad circuits up-front. For every ε, $c_0 = (T-BN)/T, \tau > 0$, if $N > i_0 \Big/ \left(B/(1-c_0) - \frac{B}{1-(1-\tau)c_0}\right)$ and $b > -(\lceil \log \varepsilon \rceil + 1) \Big/ \log((1-\tau)c_0 + \frac{B}{B/(1-c_0)-i_0/N})$, then*

$$\Pr_{fail}(N, B, T, \tau, b) < \varepsilon.$$

Last, we also considered employing mixed strategies for BatchedCut protocols (i.e., fixing T to some public value up-front while randomizing the selection of B) to further reduce the cost. However, our analysis show that the extra improvement brought by randomized strategies is very small in this setting. This is consistent with our intuition: (1) It only makes sense to alternate B between two consecutive integers, which can be derived as a corollary of [**?**, Lemma 9]; (2) The strategy with smaller B is almost dominated by the one with larger B such that mixing them brings little extra benefit. Therefore, we opt to avoid using randomized strategies for BatchedCut protocols.

Results. Figure 8 depicts the improvements induced by the refined cost model for cut-and-choose. In this scenario, our search algorithm is able to identify the optimal

```
Input: $\varepsilon, N, c_0$.
Output: $T^*, B^*$
1. Choose $b^u$ and $B^u$.
   (a) Let $s = \lceil \log \varepsilon \rceil$. Compute $i_0 := s+2$ and $b^u := -(s+1) \Big/ \log(\frac{c_0}{2} + \frac{2}{2/(1-c_0)-i_0/N})$
   (b) Set $T^* := \infty$ and $B^u := \infty$.
2. For $B$ ranging from 2 to $B^u$,
   (a) Precompute $\Pr_e(N,B,b) = 1 - f_b(0)/f_b(1)$ for every $b \in [B, b^u]$.
   (b) Find the smallest $T$, call it $T_B$, such that $T < T^*$ and $\Pr_{fail}(N,B,T_B,\tau,b) < \varepsilon$ for all $b \leq b^u$, using exponential
       backoff and binary search. (Note that $\Pr_{fail}(N,B,T,\tau,b)$ monotonically decreases with $T$ while the cost $T +$
       $NB(r-1)$ monotonically increases with $T$.)
   (c) If $T^* + NB^*(r-1) > T_B + NB(r-1)$, then update $T^* := T_B$, $B^* := B$, and $B^u := \lceil (1-c_0)T_B/N \rceil$.
3. Output $T^*, B^*$.
```

Figure 7: Find cost-efficient (T,B) for BatchedCut protocols.

pure strategies for r up to 10^5, assuming the check rate c is always larger than 0.02. We note that the optimal strategy (characterized by (T,B) pairs) does not change much for $10^5 < r \ll \infty$. Experimental results show that roughly $20 \sim 80\%$ performance gain can be achieved (while the exact improvement depends on r, ε and N). Note the effects of the bad circuits detection rate τ on the benefits of our approach (through comparing Figure 8a and 8b).

Changes to Existing Protocols. In this case, because we do not use randomized strategies, our proposal applied to the BatchedCut scenario requires *no protocol changes* other than setup the public parameters to the suggested value output by our search algorithm.

4 Benefits in Time

Recall that circuit checking will result in negligible network traffic because only a short circuit seed and a circuit hash needs to be transferred. This gap in bandwidth overhead also leads to a substantial gap in execution speeds, due to the significant difference in the throughputs of garbling/checking (less than 10 ns/gate on a single-core processor) and that of network transmission (typically 10–1000 Mbps).

To evaluate the benefit of our technique in terms of time, we modified OblivC [?] to measure the ratios of speed for circuit evaluation and circuit checking tasks in various network settings. Our test implementation utilizes Intel AES-NI instructions and the half-gate garbling technique [ZRE15] to minimize bandwidth usage, and SHA256 implementation provided by Libgcrypt for circuit hashing. In circuit checking tasks, the circuit generator garbles a number of circuits but sends only a $(Seed, Hash)$ pair for each garbled circuit, while the circuit verifier re-computes the hash from the seed for each circuit. We record the per circuit time cost for this task as T_c. For circuit evaluation tasks, the circuit generator garbles a number of circuits and sends the garbled gates to the evaluator, who not only evaluates, but also computes the hash of the received circuit. We record the per circuit time cost for this task as T_e. (In both tasks, the two ends work in a pipelined fashion.) Thus, the time cost ratios between evaluation and checking can be calculated as $(T_c + T_e)/(2T_c)$ (recall that every circuits will be generated twice, once for commiting their hashes and once for check/evaluate).

We used a benchmark circuit (provided by OblivC) with 31×10^6 non-free gates. Our exepriments were run on Amazon EC2 boxes (instance type: `c4.large`, \$0.105/hour, Intel Xeon E5-2666, 2.9GHz, 3.75GB memory) with Ubuntu 14.04 Server edition in the VA region.

We detail our experimental results in Table 5. The r values (in terms of wall-clock time) range from a little over 1 (single-core processors running over high speed connection) to several hundreds (multi-core processors over ordinary home-to-home connections). Such time gaps can be well-explained by the difference in the throughputs of computation and communication. The observed speedups of the proposed cut-and-choose technique can range from 12% up to 258%. Note that our approach yields no noticeable time savings for the settings of running SingleCut or BatchedCut protocols in a 1 Gbps LAN with single-core processors (compared to the their state-of-the-art couterparts), because the cost ratio r is already very close to 1.

We note that due to the use of SHA256 (for computing circuit hashes), we observe only 2.23×10^6 gates/second for circuit verification while 1.30×10^6 gates/second for circuit evaluation. It would be interesting to replace SHA256 with some hashing algorithm that leverages AES-NI instructions to match up with the speed of AES-NI based garbling (more than 10^9 gates/second, as was reported in [BHKR13]). That will imply a time ratio up to 100× larger than we observe in our experiments.

		LAN 1 Gbps	WAN 100 Mbps	WAN 10 Mbps	16-core 100 Mbps WAN	16-core 10 Mbps WAN
	T_e (seconds)	24.1	103.5	818	81	795
	T_c (seconds)	13.9	13.9	13.9	0.87	0.87
	r	1.37	4.22	29.9	47	457
Speedup	MajorityCut	12%	26%	106%	132%	258%
	SingleCut	0%	13%	76%	97%	211%
	BatchedCut	0%	14%	41%	47%	59%

Table 5: Timing gaps between circuit evaluation and verification and our speedup benefits (measurements taken from 10 runs with 0.1% relative standard deviation) for a 31m gate circuit.

5 Conclusion

The state-of-the-art design of cut-and-choose protocols considers an overly simplified cost model, and does not exploit the opportunity of dynamically variating e to thwart cheating adversaries. We have shown, through experiments, the dramatic gap in the bandwidth costs between circuit evaluation and circuit verification. We revisit the cut-and-choose protocol design problem in a refined cost model and give three highly efficient solvers, one for each class of cut-and-choose protocols, that output the best strategy for a particular cost ratio in our model. Simulation results show that our approach bring significant savings in bandwidth cost, as well as substantial speedups (especially when running secure computation protocols outside idealized laboratory environments). Most importantly, the benefits require very small changes to existing protocols and come completely from formal proofs that do not depend on any unprovable assumptions.

6 Acknowledgments

We thank Yuan Zhou (MIT) and Zhilei Xu (MIT) for inspiring discussions on cut-and-choose. We appreciate Xiao Wang (Maryland)'s advice that helped to benchmark state-of-the-art garbled circuit implementations. Work of Ruiyu Zhu and Yan Huang was supported by NSF award #1464113. Work of Jonathan Katz was supported in part by NSF award #1111599. Work of shelat was supported in part by NSF grants TC-0939718, TC-1111781, and 1618559, a Microsoft Faculty Fellowship, and a Google Faculty Research Award.

References

[1] Circuits of Basic Functions Suitable For MPC and FHE. http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/.

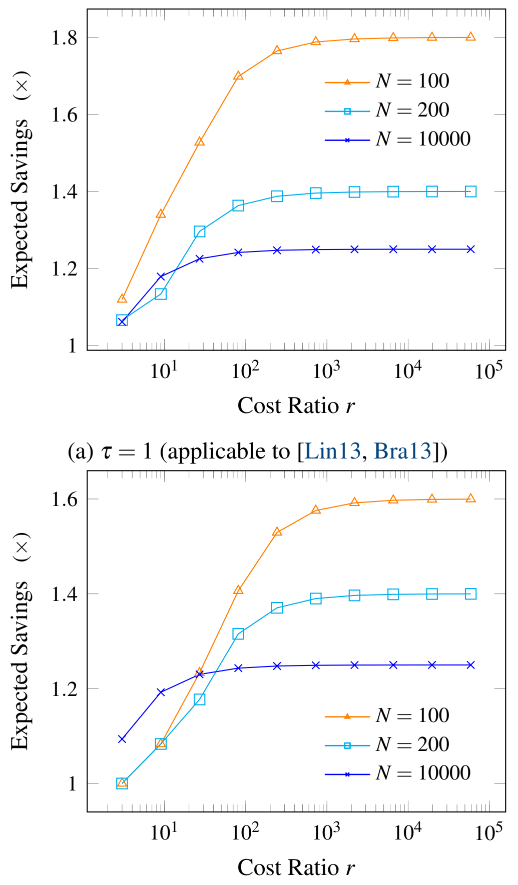
[2] A. Afshar, Z. Hu, P. Mohassel, and M. Rosulek. How to efficiently evaluate RAM programs with malicious security. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 702–729, Sofia, Bulgaria, Apr. 26–30, 2015. Springer, Heidelberg, Germany.

[3] A. Afshar, P. Mohassel, B. Pinkas, and B. Riva. Non-interactive secure computation based on cut-and-choose. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 387–404, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[4] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492, Berkeley, California, USA, May 19–22, 2013. IEEE Computer Society Press.

[5] L. 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 K. Sako and P. Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 441–463, Bengalore, India, Dec. 1–5, 2013. Springer, Heidelberg, Germany.

[6] N. Buescher and S. Katzenbeisser. Faster secure computation through automatic parallelization. In *USENIX Security Symposium*, Aug. 2015.



(a) $\tau = 1$ (applicable to [Lin13, Bra13])

(b) $\tau = 1/2$ (applicable to [**?**])

Figure 8: Bandwidth savings for BatchedCut protocols ($\varepsilon = 2^{-40}$ and the check ratio $c \geq 0.02$)

[7] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, P. S. Nordholt, and C. Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 537–556, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[8] V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In N. P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 289–306, Istanbul, Turkey, Apr. 13–17, 2008. Springer, Heidelberg, Germany.

[9] S. Gureron, Y. Lindell, A. Nof, and B. Pinkas. Fast garbling of circuits under standard assumptions. In *Conference on Computer and Communications Security*, 2015.

[10] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[11] Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 18–35, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Heidelberg, Germany.

[12] Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J. Malozemoff. Amortizing garbled circuits. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 458–475, Santa Barbara, CA, USA, Aug. 17–21, 2014. Springer, Heidelberg, Germany.

[13] Y. Huang and R. Zhu. Revisiting LEGOs: Optimizations, analysis, and their limit. Cryptology ePrint Archive, Report 2015/1038, 2015. `http://eprint.iacr.org/2015/1038`.

[14] N. Husted, S. Myers, A. Shelat, and P. Grubbs. Gpu and cpu parallelization of honest-but-curious secure two-party computation. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, pages 169–178, New York, NY, USA, 2013. ACM.

[15] B. Kreuter, A. Shelat, and C. hao Shen. Billion-gate secure computation with malicious adversaries. In *The 21st USENIX Security Symposium (USENIX Security 12)*, pages 285–300, Bellevue, WA, 2012. USENIX.

[16] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Heidelberg, Germany.

[17] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In M. Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany.

[18] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology*, 25(4):680–722, Oct. 2012.

[19] Y. Lindell, B. Pinkas, and N. P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *SCN 08: 6th International Conference on Security in Communication Networks*, volume 5229 of *Lecture Notes in Computer Science*, pages 2–20, Amalfi, Italy, Sept. 10–12, 2008. Springer, Heidelberg, Germany.

[20] Y. Lindell and B. Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 476–494, Santa Barbara, CA, USA, Aug. 17–21, 2014. Springer, Heidelberg, Germany.

[21] Y. Lindell and B. Riva. Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In *Computer and Communication Security (CCS)*, 2015.

[22] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi. ObliVM: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy*, pages 359–376, San Jose, California, USA, May 17–21, 2015. IEEE Computer Society Press.

[23] P. Mohassel and B. Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 36–53, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Heidelberg, Germany.

[24] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. GraphSC: Parallel secure computation made easy. In *2015 IEEE Symposium on Security and Privacy*, pages 377–394, San Jose, California, USA, May 17–21, 2015. IEEE Computer Society Press.

[25] J. B. Nielsen and C. Orlandi. LEGO for two-party secure computation. In O. Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 368–386. Springer, Heidelberg, Germany, Mar. 15–17, 2009.

[26] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In M. Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, Tokyo, Japan, Dec. 6–10, 2009. Springer, Heidelberg, Germany.

[27] A. Shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 386–405, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[28] A. Shelat and C.-H. Shen. Fast two-party secure computation with minimal assumptions. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 523–534, Berlin, Germany, Nov. 4–8, 2013. ACM Press.

[29] D. P. Woodruff. Revisiting the efficiency of malicious two-party computation. In M. Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 79–96, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany.

[30] S. Zahur and D. Evans. Obliv-c: A language for extensible data-oblivious computation.

[31] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250, Sofia, Bulgaria, Apr. 26–30, 2015. Springer, Heidelberg, Germany.

A Proof of Claim 1

Proof There exists N_0 such that if $N > N_0$,

$$\begin{aligned}
\frac{\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} &= \frac{(T-b)!(T-BN)!(BN)!}{T!(T-BN-i)!(BN-b+i)!} \\
&= \frac{((T-BN-i+1)\cdots(T-BN))((BN-b+i+1)\cdots BN)}{(T-b+1)\cdots T} \\
&= \frac{(T-BN-i+1)\cdots(T-BN)}{(T-i+1)\cdots T}\cdot\frac{(BN-b+i+1)\cdots BN}{(T-b+1)\cdots(T-i)} \\
&\le \left(\frac{T-BN}{T}\right)^i\left(\frac{BN}{T-i}\right)^{b-i} \triangleq U.
\end{aligned}$$

Similarly, we have, there exists N_1 such that if $N > N_1$,

$$\frac{\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \ge \left(\frac{T-BN-i+1}{T-i+1}\right)^i\left(\frac{BN-b+i+1}{T-b+1}\right)^{b-i} \triangleq L.$$

So, we know that, for sufficiently large N,

$$\begin{aligned}
U\Bigg/\frac{\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} &\le \frac{U}{L} = \left(\frac{T-BN}{T-BN-i+1}\cdot\frac{T-i+1}{T}\right)^i\left(\frac{BN}{BN-b+i+1}\cdot\frac{T-b+1}{T-i}\right)^{b-i} \\
&= \left[\frac{(T-BN)T-(i-1)T+(i-1)BN}{(T-BN)T-(i-1)T}\right]^i\cdot \\
&\qquad\left[\frac{(BN-b+i+1)(T-i)+(b-i-1)(T-BN-i)}{(BN-b+i+1)(T-i)}\right]^{b-i} \\
&= \left[1+\frac{(i-1)BN}{(T-BN)T-(i-1)T)}\right]^i\left[1+\frac{(b-i-1)(T-BN-i)}{(BN-b+i+1)(T-i)}\right]^{b-i} \\
&\le \left(1+\frac{i-1}{T-BN-i+1}\right)^i\left(1+\frac{b-i-1}{BN-b+i+1}\right)^{b-i} \qquad (4) \\
&\le \left(1+\frac{i-1}{T-BN-i+1}\right)^i\left(1+\frac{b-1}{BN-b+1}\right)^{b} \qquad (5)
\end{aligned}$$

Note that the inequality (4) holds because $T > BN$. Thanks to the upper-bound of b (Claim 2) and hence on i (recall $i \le b$), $\lim_{N\to\infty}\left(1+\frac{i-1}{T-BN-i+1}\right)^i\left(1+\frac{b-1}{BN-b+1}\right)^b = 1$. ■

A.1 Proof of Claim 2

Proof Let $0 < \tau \le 1$ be the probability that P_2 detects the abnormality in checking garbled gate g conditioned on g is indeed bad. We have

$$\Pr_{fail}(N,b) = \sum_{i=0}^{b}(1-\tau)^i\frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}}\Pr_e(N,b-i)$$

where $(1-\tau)^i\binom{b}{i}\binom{T-b}{T-BN-i}\Big/\binom{T}{T-BN}$ is the probability that P_1 who generates b bad gates survives the gate verification stage with i bad gates selected for verification (but P_2 fails to detect any of them). Because there exists i_0 such that

$(1-\tau)^{i_0} < \varepsilon/2$,

$$\begin{aligned}
\Pr_{fail}(N,b) &= \sum_{i=0}^{b} (1-\tau)^i \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \Pr_e(N,b-i) \\
&= \sum_{i=0}^{i_0} (1-\tau)^i \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \Pr_e(N,b-i) + \sum_{i=i_0+1}^{b} (1-\tau)^i \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \Pr_e(N,b-i) \\
&\le \sum_{i=0}^{i_0} (1-\tau)^i \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \Pr_e(N,b-i) + (1-\tau)^{i_0} \sum_{i=i_0+1}^{b} \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \Pr_e(N,b-i) \\
&\le \sum_{i=0}^{i_0} (1-\tau)^i \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \Pr_e(N,b-i) + \frac{\varepsilon}{2} \sum_{i=i_0+1}^{b} \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \\
&\le \sum_{i=0}^{i_0} (1-\tau)^i \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \Pr_e(N,b-i) + \frac{\varepsilon}{2} \sum_{i=1}^{b} \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \\
&\le \sum_{i=0}^{i_0} (1-\tau)^i \frac{\binom{b}{i}\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \Pr_e(N,b-i) + \frac{\varepsilon}{2} \cdot 1 \\
&\le \sum_{i=0}^{i_0} (1-\tau)^i \binom{b}{i} \left(\frac{T-BN}{T}\right)^i \left(\frac{BN}{T-i}\right)^{b-i} \Pr_e(N,b-i) + \frac{\varepsilon}{2} \qquad \text{[Claim 3]} \\
&\le \sum_{i=0}^{i_0} (1-\tau)^i \binom{b}{i} \left(\frac{T-BN}{T}\right)^i \left(\frac{BN}{T-i}\right)^{b-i} \Pr_e(N,b) + \frac{\varepsilon}{2} \\
&\le \sum_{i=0}^{i_0} (1-\tau)^i \binom{b}{i} \left(\frac{T-BN}{T}\right)^i \left(\frac{BN}{T-i_0}\right)^{b-i} \Pr_e(N,b) + \frac{\varepsilon}{2} \\
&\le \sum_{i=0}^{b} (1-\tau)^i \binom{b}{i} \left(\frac{T-BN}{T}\right)^i \left(\frac{BN}{T-i_0}\right)^{b-i} \Pr_e(N,b) + \frac{\varepsilon}{2} \\
&= \left((1-\tau)\frac{T-BN}{T} + \frac{BN}{T-i_0}\right)^b \Pr_e(N,b) + \frac{\varepsilon}{2} \\
&\le \left((1-\tau)\frac{T-BN}{T} + \frac{BN}{T-i_0}\right)^b + \frac{\varepsilon}{2}.
\end{aligned}$$

Thus, $N > i_0 \Big/ \left(\frac{B}{1-c_0} - \frac{B}{1-(1-\tau)c_0}\right)$ ensures that $(1-\tau)\frac{T-BN}{T} + \frac{BN}{T-i_0} < 1$, while $b > -(s+1) \Big/ \log((1-\tau)c_0 + \frac{B}{B/(1-c_0)-i_0/N})$ ensures that $\left((1-\tau)\frac{T-BN}{T} + \frac{BN}{T-i_0}\right)^b + \frac{\varepsilon}{2} \le 2^{-s}$. Therefore we can conclude that $\Pr_{fail}(N,B,T,\tau,b) < \varepsilon$. ■

Claim 3 *If T,N,b,i are non-negative integers such that $T > BN$, $T \ge b$, and $i \le b$, then*

$$\frac{\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} \le \left(\frac{T-BN}{T}\right)^i \left(\frac{BN}{T-i}\right)^{b-i}.$$

Proof

$$\begin{aligned}
\frac{\binom{T-b}{T-BN-i}}{\binom{T}{T-BN}} &= \frac{(T-b)!(T-BN)!(BN)!}{T!(T-BN-i)!(BN-b+i)!} \\
&= \frac{\big[(T-BN-i+1)\cdots(T-BN)\big]\big[(BN-b+i+1)\cdots BN)\big]}{(T-b+1)(T-b+2)\cdots T} \\
&= \frac{(T-BN-i+1)\cdots(T-BN)}{(T-i+1)\cdots T} \cdot \frac{(BN-b+i+1)\cdots BN}{(T-b+1)\cdots(T-i)} \le \left(\frac{T-BN}{T}\right)^i \left(\frac{BN}{T-i}\right)^{b-i}
\end{aligned}$$

■