Cost-Aware Cut-and-Choose Games with Applications in Cryptography and Prefix-Free Codes

Ruiyu Zhu and Yan Huang

Indiana University, Bloomington
{zhu52,yh33}@indiana.edu

February 21, 2017

Abstract

Cost-aware cut-and-choose game is a fundamental technique that has many cryptographic applications. Best existing solutions of this game assumed for simplicity that the number of challenges is publicly known. This paper considers an extension of this game where the number of challenges can be picked *probabilistically* and hidden to the adversary. Although this small change leads to a linear program with infinitely many variables and constraints, we discover a surprising efficiency solver — using only $O(n^2)$ space and $O(n^2)$ time where n is the upper-bound on the number of challenges allowed in the strategy space. We then prove that n is bounded for any fixed cost ratio, implying the optimality of our solution extends to the strategy space that allow any number of challenges. As two interesting applications of our game solver, we demonstrate its value in constructing an actively secure two-party computation protocol and an optimal prefix-free code for encryptions.

1 Introduction

Consider a two-step probabilistic game $\mathsf{Cut\text{-}and\text{-}Choose}_{\mathcal{A},\mathcal{C}}(t, r, \varepsilon)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

1. $\mathcal{A}$ creates and sends t boxes. Each of the t boxes is either *empty* or *filled*.
2. $\mathcal{C}$ (without looking at any box) arranges the t boxes into two groups, S_0 and S_1. $\mathcal{C}$ pays $|S_0| + r \cdot |S_1|$ units of cost for her arrangement (where $|S_0|$ and $|S_1|$ are the sizes of S_0 and S_1, and r is a positive constant parameter of the game).

Per outcome, $\mathcal{A}$ wins the game *if and only if* every box in S_0 is filled meanwhile every box in S_1 is empty. It can be regarded as a *zero-sum* game where the *utility* is the winning odds. However, unlike traditional treatment of zero-sum games, here we introduce a metric of *cost* that is relevant to one of the two players (i.e., $\mathcal{C}$) and aim to study optimal strategies achieve certain utility while minimize the cost. Note that the utility and the cost are two separate metrics that never mix up. E.g., the $|S_0| + r \cdot |S_1|$ units of cost $\mathcal{C}$ pays at step 2 should not be counted into $\mathcal{A}$'s utility gain.

Interesting questions about this game include, *What is the minimum expected cost for $\mathcal{C}$ to guarantee her winning odd of at least $1 - \varepsilon$? How should $\mathcal{C}$ play to achieve this?* Their equivalent dual questions are, *Given a particular budget, what is the maximum probability that $\mathcal{C}$ can win? How should $\mathcal{C}$ play to achieve this?*

1. $\mathcal{A}$ creates and sends boxes that are either *empty* or *filled* until $\mathcal{C}$ tells him to stop.
2. $\mathcal{C}$ samples an integer t and instructs $\mathcal{A}$ to stop right after t boxes are received. $\mathcal{C}$ arranges the t boxes into two groups, S_0 and S_1, and pays $|S_0| + r \cdot |S_1|$ units of cost (where $|S_0|$ and $|S_1|$ are the sizes of S_0 and S_1).

Outcome: $\mathcal{A}$ wins the game *if and only if* every box in S_0 is filled meanwhile every box in S_1 is empty.

Figure 1: The extended cut-and-choose game $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}(r, \varepsilon)$

While this game can be efficiently solved through mapping it into a continuous backpacking problem [23], we consider, in this paper, an extension of the above game where the total number of challenges, t, will be randomized and $\mathcal{A}$ learns t only *after* t boxes are prepared and sent. We formalize the extended cut-and-choose game in Figure 1. While this small change offers $\mathcal{C}$ a better chance to win, it raises several more challenging questions:

1. *Can we precisely quantify the benefit of $\mathcal{C}$ due to allowing t to be randomized?*
2. *How should $\mathcal{C}$ pick t to maximize her benefits? Does it make sense to allow $t \to \infty$?*
3. *What is $\mathcal{C}$'s optimal game play?*

Practical values of these questions will be evident in concrete applications given below.

1.1 Motivation

Solutions to the extended cut-and-choose game $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}$ have applications in two very different contexts. First, it can be used to force honest behavior in executing cryptographic protocols. Second, the idea leads to an efficient prefix-free *source coding* scheme designed to encode encryptions. We will give the mathematical models for both problems and show their equivalence in Section 2.

Forcing Honest Behavior. Cut-and-choose serves an important technique to ensure security of cryptographic protocols against malicious adversaries who can deviate from the protocol in arbitrary ways to violate security. To force honest behavior, the idea is to first require the potential adversary to repetitively run its protocol steps (with a fresh random tape every time); then only a subset of the runs will be used to accomplish the protocol while the rest are checked to verify honest behavior. In this context, $\mathcal{A}$ "preparing a filled box" corresponds to "running the cryptographic protocol honestly" but "preparing an empty box" corresponds to "deviating from the prescribed protocol"; while $\mathcal{C}$ "placing a box in S_0" corresponds to "checking" the execution and "placing a box in S_1" corresponds to "using" the execution to accomplish the cryptographic protocol. Moreover, the cryptographic protocol is designed such that a security failure corresponds precisely to "$\mathcal{A}$ wins the cut-and-choose game", whose probability will be bounded by a known parameter ε. Arranging the boxes into two groups is analogous to challenging the adversary with one of two choices, for which a cheating adversary can't properly respond in both ways. We note that the cost ratio parameter r reflects that the costs between the two ways to challenge the adversary can be different (i.e. when $r \neq 1$).

The cut-and-choose technique has been used in protocols for fair exchange of digital currency [2], zero-knowledge proofs [5], secure delegation of computation [9], and secure two-party computation [18, 15, 7, 1]. To appreciate the cost gaps between different types of challenges, we first look at the

Zero-Knowledge Proof of Knowledge protocol for Hamiltonian cycles [5] as an example. In this protocol, the parties need to de-commit and verify either a matrix ($|V|^2$ items) representation of an isomorphic graph or a specific Hamiltonian cycle ($|V|$ items), where $|V|$ denotes the number of vertices in the graph, hence $r = |V|$ (recall that $|V|$ has to be large enough for security to hold). In secure computation settings, researchers have also realized that the cost ratios are affected by many factors including the choice of cryptographic primitives and the software/hardware environment, thus can variate in a wide range (e.g., 10–1000) in practice [1, 23].

Cut-and-choose was first applied to cryptographic protocol design without taking into account the cost difference in different types of challenges [18, 15, 7, 2, 9]. More recently, Zhu et al. [23] presented a cost-aware, game-theoretical analysis of cut-and-choose games and demonstrated a 1.1–3.5x speedup (for r ranging from 4 to 1024) using some mixed strategies. However, they assumed for simplicity that the total number of challenges (i.e., t) is publicly fixed in advance so that the best strategy for $\mathcal{C}$ can be found by solving a linear program of only t variables and $2t + 1$ constraints.

However, in these protocols, the total number of challenges, t, does not need to be *publicly* fixed in advance. Instead, t can be picked *secretly* and revealed to $\mathcal{A}$ only *after* t challenge-responses are done (though, in one round). Intuitively, this enhancement offers $\mathcal{C}$ additional advantage. Unfortunately, solving such games requires solving an extended linear program with *infinitely* many variables and *infinitely* many constraints! Note that even finding an sub-optimal strategy assuming $t < n$ for a fixed n can be difficult, as there will be $O(2^n)$ pure strategies to choose from and the resulting linear program still has exponentially many variables and constraints. Thus, *a priori*, the problem seems exponentially hard.

Optimal Prefix-free Codes of Encryptions. Suppose one would like to use a *prefix-free* code to transmit a long stream of encryptions. Prefix-free code, also known as *instantaneous code*, is a variable-length code in which no valid codeword can be a prefix of any other valid codeword. It allows efficient one-pass decoding without needing any "out-of-band" special markers between codewords. For regular messages of a known distribution, Huffman code can be used to generate prefix-free codewords in a way that reduces the overall length of the encoded message.

We aim at achieving a similar effect as what Huffman code brings to regular messages, but in a setting distinguished by two modifications:

1. The input is a stream of encryptions, hence, by security definition, the symbols always appear *uniformly*-distributed regardless of the symbol-length. (So Huffman code won't help to reduce the cost here.)

2. The cost differs between handling a 0 signal and a 1 signal. But we aim to minimize the overall *cost* of handling the encoded message. (In contrast, traditional Huffman code assumes metric where a 0 costs the same as a 1.)

This special setting is well-motivated by many real-world applications. On the one hand, the ubiquitous use of encrypted channels leaves most of the network (or storage) devices only having access to encryptions, which makes existing prefix-free codes inapplicable. Secondly, it is not uncommon to see cost metrics where handling a 0 differs considerably from that of a 1. Cost metrics of this property can be *time* (like wtih Morse code or certain Solid-State Drives technologies [20, 4, 17], transmitting/writing a 1 takes more time than a 0), *energy* (since 1 would be represented by a high voltage/frequency signal whereas 0 by a less costly low voltage/frequency signal), or even *error probability* (since, with some media, 1s are more susceptible to interference than 0s, hence more costly). Therefore, our goal is to construct a prefix-free code with minimal overall cost.

The more general topic of unequal-cost Huffman codes, which does not constrain the input symbols to appear in uniform distribution like we do, has been intensively studied in the theory

literature [13, 11, 14, 6, 16]. However, this work approaches the problem from a unique angle — focusing on uniformly distributed message-encryptions and gives practically highly efficient encode/decode procedures that are proven to output optimal prefix-free codes.

1.2 Contribution

We model and solve the extended cut-and-choose game (Figure 1). Although this optimization problem appears to involve an *infinite* number of variables and constraints, we identify an $O(n^2)$-space, $O(n^2)$-time solver where n is the upper-bound of t. Since we prove that n can always be bounded in an optimal strategy, solutions output by our solver are also optimal when compared to strategies that allow $t \to \infty$ (Claim 10).

As a first application of $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}$, we propose a cut-and-choose-based *constant-round* actively-secure two-party computation protocol that incorporates the concrete strategies output by our solver. In our protocol, the circuit generator (the $\mathcal{A}$) keeps sending garbled-circuit-hashes (the boxes) until being notified to stop through an asynchronous channel. Thus, no additional rounds are needed. We formally present the protocol in Section G.2 and prove its security in the ideal/real model paradigm (Section G.3). Given the same budget, our technique can thwart active attacks 2x better than best existing work [23].

As a second application of $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}$, we propose an optimal prefix-free code for encryptions which exploits the cost differences between transmitting/writing 0s and 1s. Potential applications of this code include storage controllers on encrypted SSDs and network transceivers handling encrypted traffic.

2 Problem Statement

Mathematical Model. The extended cut-and-choose game $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}(r,\varepsilon)$ is formalized in Figure 1. Let n be an upper-bound of t. Fixing n, we denote a pure strategy of $\mathcal{A}$ by an n-bit binary string, $\mathsf{a} = \mathsf{a}_1 \dots \mathsf{a}_n$, where $\mathsf{a}_i = \mathtt{0}$ denotes making the i^{th} box *filled* whereas $\mathsf{a}_i = \mathtt{1}$ denotes making the i^{th} box *empty*.[1] Symmetrically, $\mathcal{C}$'s strategy can be denoted by a binary string of length t ($t \le n$), $\mathsf{c} = \mathsf{c}_1 \dots \mathsf{c}_t$, where $\mathsf{c}_i = \mathtt{0}$ denotes $\mathcal{C}$ puts the i^{th} box to S_0 whereas $\mathsf{c}_i = \mathtt{1}$ denotes $\mathcal{C}$ puts the i^{th} box to S_1. Thus, $\mathcal{A}$'s strategy $\mathsf{a} = \mathsf{a}_1 \dots \mathsf{a}_n$ wins a $\mathcal{C}$'s strategy $\mathsf{c} = \mathsf{c}_1 \dots \mathsf{c}_t$ ($t \le n$) if and only if $\mathsf{a}_i = \mathsf{c}_i$ for all $i \in \{1, \dots, t\}$, namely, c is a prefix of a. For example, if $n = 5$, $\mathsf{a} = \mathtt{10110}$ wins all strategies $\mathsf{c} \in \{\mathtt{1}, \mathtt{10}, \mathtt{101}, \mathtt{1011}, \mathtt{10110}\}$.

Let x_{a} be the probability that $\mathcal{C}$ groups the boxes according to a binary string a. Let $\mathsf{zeros}(\mathsf{a})$ and $\mathsf{ones}(\mathsf{a})$ be the numbers of 0s and 1s in a, e.g. $\mathsf{zeros}(\mathtt{01011}) = 2$, $\mathsf{ones}(\mathtt{01011}) = 3$. Let $\mathsf{len}(\mathsf{a})$ be the length of a. Let $\mathsf{cost}(\mathsf{a}) = \mathsf{zeros}(\mathsf{a}) + r \cdot \mathsf{ones}(\mathsf{a})$.

Fixing n, the cut-and-choose game for a given ε and r can be modeled by the mathematical program described in Figure 2. Note that the equality constraint (3) reflects that $\{\mathsf{a} \mid \mathsf{a} \in \{\mathtt{0},\mathtt{1}\}^t, 1 \le t \le n\}$ is a probability space. The constraint (4) expands to 2^n individual inequalities, one for each string in $\{\mathtt{0},\mathtt{1}\}^n$. It reflects the requirement that no matter how $\mathcal{A}$ chooses his strategy a, his winning odds is at most ε. This linear program has exponentially many (in terms of n) variables (an x_{a} for every $\mathsf{a} \in \{\mathsf{a} \mid \mathsf{a} \in \{\mathtt{0},\mathtt{1}\}^t, 1 \le t \le n\}$, and exponentially many constraints collectively written as constraint (4). For a practically useful n ($n > 20$), it is computationally infeasible to solve this linear program with black-box calls to state-of-the-art LP solvers.

Literally, solving $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}(r,\varepsilon)$ requires solving an array of instances of the above mathematical program for all $n \le \infty$.

[1] We use Sans Serif letters to denote bit-string variables and fixed-width letters to denote bit variables or bit values.

$$\min \sum x_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a})$$

subject to

$$\mathsf{len}(\mathsf{a}) \leq n \tag{1}$$

$$x_{\mathsf{a}} \geq 0, \quad \forall \mathsf{a} \in \{0,1\}^t, 1 \leq t \leq n \tag{2}$$

$$\sum x_{\mathsf{a}} = 1 \tag{3}$$

$$x_{\mathsf{a}_1} + x_{\mathsf{a}_1\mathsf{a}_2} + \cdots + x_{\mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n} \leq \varepsilon, \quad \forall \mathsf{a} = \mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n \in \{0,1\}^n \tag{4}$$

Figure 2: Modeling the cut-and-choose game for a fixed n. Due to symmetry, we can assume, without loss of generality, that $r \geq 1$. The linear program involves $2^{n+1} - 2$ variables, i.e., an x_{a} for every $\mathsf{a} \in \{\mathsf{a} \mid \mathsf{a} \in \{0,1\}^t, 1 \leq t \leq n\}$. The cost of cut-and-choose with each a is $\mathsf{cost}(\mathsf{a}) = \mathsf{zeros}(\mathsf{a}) + r \cdot \mathsf{ones}(\mathsf{a})$. The equality constraint (3) reflects the fact that $\{\mathsf{a} \mid \mathsf{a} \in \{0,1\}^t, 1 \leq t \leq n\}$ is a probability space. The constraint (4) expands to 2^n inequalities (one for each string in $\{0,1\}^n$), reflecting the requirement that however $\mathcal{A}$ chooses a, his winning odds is at most ε.

Modeling The Prefix-free Encoding Problem. In this context, we assume the alphabet for the input ciphertext contains 2^k k-bit symbols and the alphabet for the output encoding has 2^t variable length symbols. Assuming the ciphertext is produced with a cryptographically-secure encryption scheme, then every input symbol occurs with equal probability in the ciphertext, i.e., $1/2^k$. Borrowing the len and cost functions defined above and let a denote a codeword, hence $\mathsf{len}(\mathsf{a}) \leq t$ and the cost associated with a is $\mathsf{cost}(\mathsf{a})$. Thus, the expected cost per codeword is $\sum x_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a})$ where $x_{\mathsf{a}} \in \{0, 1/2^k\}$ for all a. Upper-bound t by n, the mathematical program for computing the minimal cost of the prefix-free code is

$$\min \sum x_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a})$$

subject to

$$\mathsf{len}(\mathsf{a}) \leq n$$

$$x_{\mathsf{a}} \geq 0, \quad \forall \mathsf{a} \in \{0,1\}^t, 1 \leq t \leq n$$

$$\sum x_{\mathsf{a}} = 1$$

$$x_{\mathsf{a}_1} + x_{\mathsf{a}_1\mathsf{a}_2} + \cdots + x_{\mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n} \leq 2^{-k}, \quad \forall \mathsf{a} = \mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n \in \{0,1\}^n$$

$$x_{\mathsf{a}} \in \{0, 2^{-k}\}, \quad \forall \mathsf{a} \in \{0,1\}^t, 1 \leq t \leq n.$$

The "prefix-free" property is guaranteed by the combination of the last three constraints, so for any length n string, only one of its n prefixes can be a valid codeword. By solving the above program, we can identify exactly 2^k *ordered* codewords whose associated x_{a} are non-zero. An optimal encoding algorithm simply maps the 2^k input symbols to these 2^k codewords.

Comparing to the mathematical program of $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}$ (Figure 2), the mathematical program for prefix-free codes has the same target function and almost the same constraints, except replacing ε with 2^{-k} and (2) adding a 5^{th} constraint: $x_{\mathsf{a}} \in \{0, 2^{-k}\}, \forall \mathsf{a}$. Thus, an optimal solution for the program of Figure 1 will also be an optimal solution to the above program if the solution also satisfies the constraint that $x_{\mathsf{a}} \in \{0, 2^{-k}\}, \forall \mathsf{a}$, a fact we will prove as Claim 8. Thus, we can restrict our attention to solving $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}$.

Roadmap. We first describe an efficient solver to $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}(r, \varepsilon)$ for any fixed n (Section 3), then show how to generalize this result to the setting where $n \to \infty$ (Section 4). Finally, we quantify the benefits of our approach in the two motivating applications (Section 5).

Other Notations. Let $\mathbb{R}$ be the set of real numbers, $\mathbb{Q}$ be the set of rational numbers, and $\mathbb{Z}$ be the set of integers. Fix $r \in \mathbb{Q}$, we use $\mathbb{Z}[r]$ to denote the set $\{a + b \cdot r \mid a, b \in \mathbb{Z}\}$. For all $x \in \mathbb{R}$, we define $\langle x \rangle = (|x| + x)/2$. Unless specified otherwise, throughout the paper we denote by ε the probability that $\mathcal{A}$ wins the game, r the cost ratio between the two types of challenges, and n the fixed upper-bound on the number of challenges.

3 Solve $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}(r, \varepsilon)$ for bounded t

We will show a surprisingly efficient solver for $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}(r, \varepsilon)$. Theorem 1 states its efficiency.

Theorem 1. *Given a cost ratio $r \in \mathbb{Q}$, a security parameter $0 < \varepsilon \leq 1$, and a bound n on t, the linear program of Figure 2 can be solved using $O(n^2)$ space and $O(n^2)$ time.*

Note that it suffices to consider the case where $r \geq 1$ as the game structure is symmetric. In addition, if $n < \lceil \log(1/\varepsilon) \rceil$, then $\sum x_{\mathsf{a}} \leq \sum_{\mathsf{a}} \varepsilon \leq 2^n \varepsilon < 1$, namely, constraint (3) can't be satisfied. So it suffices to assume $n \geq \lceil \log(1/\varepsilon) \rceil$. Our proof below is constructive and suggests concrete game-playing strategies.

3.1 Intuition and Insights

Let $y_{\mathsf{a}} = \varepsilon \cdot x_{\mathsf{a}}$ and $v = 1/\varepsilon$, the original linear program (Figure 2) can be reformulated as

$$\begin{aligned} \min \quad & v \cdot \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \\ \text{subject to} \quad & \\ & \mathsf{len}(\mathsf{a}) \leq n \\ & y_{\mathsf{a}} \geq 0, \quad \forall \mathsf{a} \text{ where } \mathsf{len}(\mathsf{a}) \leq n \\ & \sum y_{\mathsf{a}} = v \\ & y_{\mathsf{a}_1} + y_{\mathsf{a}_1\mathsf{a}_2} + \cdots + y_{\mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n} \leq 1, \quad \forall \mathsf{a} = \mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n \in \{0,1\}^n. \end{aligned}$$

Because v is a constant solely determined by ε, we can leave it out from the target formula for now and focus on solving a variant linear program (Figure 3) with respect to v, r. Let $g_n(v)$ be the solution (the minimal cost) to this variant LP. If we know how to compute $g_n(v)$ efficiently given r, v, then we can solve the original LP (Figure 2) for r, ε by computing $(1/\varepsilon) \cdot g_n(1/\varepsilon)$. Hence, from this point on, we restrict our attention to the LP variant for $0 < v \leq 2^n$.[2]

$$\begin{aligned} \min \quad & \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \\ \text{subject to} \quad & \\ & \mathsf{len}(\mathsf{a}) \leq n && \text{(1 revisited)} \\ & y_{\mathsf{a}} \geq 0, \quad \forall \mathsf{a} \in \{0,1\}^t, 1 \leq t \leq n && \text{(2 revisited)} \\ & \sum y_{\mathsf{a}} = v && (5) \\ & y_{\mathsf{a}_1} + y_{\mathsf{a}_1\mathsf{a}_2} + \cdots + y_{\mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n} \leq 1, \quad \forall \mathsf{a} = \mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n \in \{0,1\}^n && (6) \end{aligned}$$

Figure 3: The variant linear program.

[2] Although we only care about solutions for $v \geq 1$ (as $\varepsilon \leq 1$), for the sake of inductive proof, it is convenient to generalize the problem to also considering $0 < v < 1$.

We will begin with finding the closed form formulas of $g_1(v)$ and $g_2(v)$ (Appendix A shows the detailed steps to derive $g_1(v)$ and $g_2(v)$):

$$g_1(v) = \begin{cases} v & 0 \le v \le 1 \\ v + (r-1)(v-1) & 1 < v \le 2. \end{cases}$$

$$g_2(v) = \begin{cases} v & 0 \le v \le 1 \\ v + (r-1)(v-1) & 1 < v \le 2 \\ v + (r-1)(v-1) + 2(v-2) & 2 < v \le 3 \\ v + (r-1)(v-1) + 2(v-2) + (r-1)(v-3) & 3 < v \le 4 \end{cases}$$

The formulas of $g_1(v)$ and $g_2(v)$ hint for a conjecture that $g_n(v)$ may be a continuous piece-wise linear functions of v for all n, with integer turning points. That is, $g_n(v) = \sum_{i=0}^{2^n-1} G_n[i] \cdot \langle v - i \rangle$ for some array G_n of constants solely determined by r ($r \in \mathbb{Q}, r \ge 1$).

Roadmap for the rest of this section. We first define a related linear program (Figure 4) and relate its solution to that of the linear program of Figure 3 by Claim 2 and Claim 3. Then, we prove that $g_n(v)$ is a continuous piece-wise linear function of v (Claim 4) by mathematical induction where Lemma 11 serves the base step of the proof and Lemma 12 and 13 provide the inductive step. Note Lemma 12 and 13 are proved using Claim 3 and 2, respectively.

Next, we show how to efficiently compute $g_n(v)$ using only $O(n)$ space. The key insight is that most entries in $g_n(v)$'s coefficient array G_n are zero (Claim 6), although G_n has 2^n entries.

Finally, we show how to derive optimal strategies for $\mathcal{C}$ to actually play the generalized cut-and-choose game. While a naïve description of the optimal strategy requires exponential space, our key idea to overcome this high demand in space is to provide an efficient algorithm (Algorithm 1) for $\mathcal{C}$ to sample her optimal strategy distribution.

3.2 Formal Analysis

We now introduce another linear program variant described in Figure 4. Compared with the LP of Figure 3, this new LP has the same target formula as the original LP of Figure 2 and considers a more restrictive strategy space—only strategies that start with `0`. Thus, the number of variables involved in this new LP is roughly halved, reducing from $2^{n+1} - 2$ to $2^n - 1$.

$$\min \quad \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a})$$

subject to

$$y_{\mathsf{a}} = 0, \quad \forall\, \mathsf{a} = 1 \| \{0,1\}^t, 0 \le t \le n-1 \qquad (7)$$

$$\mathsf{len}(\mathsf{a}) \le n \qquad (1 \text{ revisited})$$

$$y_{\mathsf{a}} \ge 0, \quad \forall\, \mathsf{a} \in \{0,1\}^t, 1 \le t \le n \qquad (2 \text{ revisited})$$

$$\sum y_{\mathsf{a}} = v \qquad (5 \text{ revisited})$$

$$y_{\mathsf{a}_1} + y_{\mathsf{a}_1\mathsf{a}_2} + \cdots + y_{\mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n} \le 1, \quad \forall\, \mathsf{a} = \mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n \in \{0,1\}^n. \qquad (6 \text{ revisited})$$

Figure 4: The linear program variant defining $f_n(v)$. ("$\|$" denotes concatenation.)

Let $g_n(v)$ be the solution to the LP of Figure 3, and $f_n(v)$ be the solution to the LP of Figure 4. Now we derive two important reductions between $g_n(v)$ and $f_n(v)$.

Claim 2. *For any $v \in \mathbb{R}^+$ and $v \leq 2^n$,*

$$g_n(v) = \min_{0 \leq u \leq v} \{\ f_n(u) + f_n(v-u) + (r-1)(v-u)\ \}.$$

For every r, we thus define $u_n(v)$ to be the u value that minimizes $g_n(v)$.

Claim 3. *For any $v \in \mathbb{R}^+$, $v \leq 2^n$ and integer $n \geq 1$,*

$$f_{n+1}(v) = v + \min_{\max\left(0,\, \frac{v-1}{2^n-1}\right) \leq t \leq 1} \left\{\ t \cdot g_n\big(\,(v+t-1)/t\,\big)\ \right\}.$$

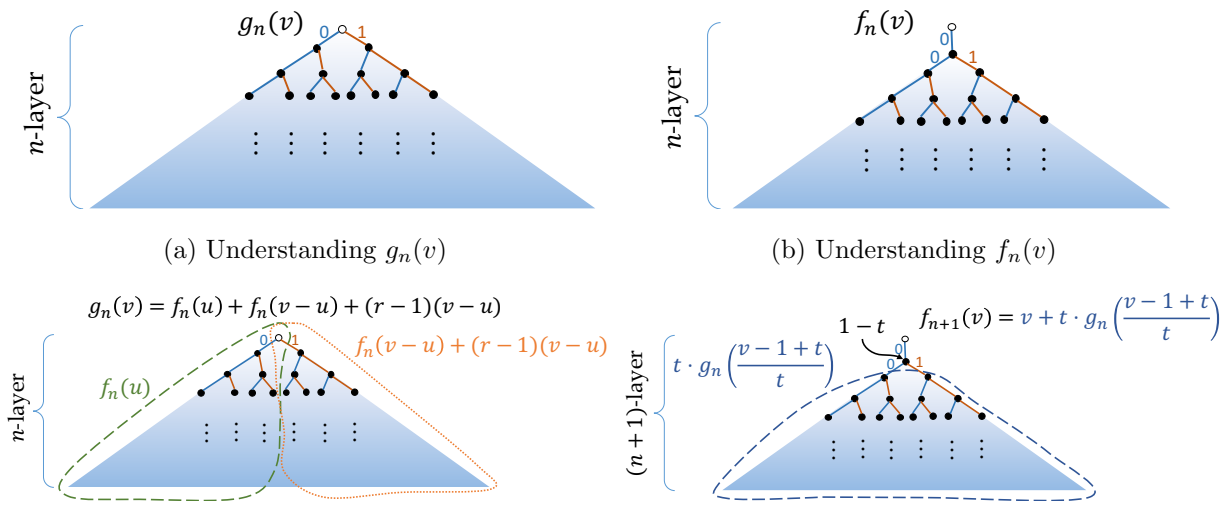
We will prove both claims above algebraically in Appendix B, but illustrate them here more intuitively with Figure 5. Envision an n-layer (numbered from 0 to $n-1$) *perfect binary tree* where all left-edges are associated with `0` and all right-edges with `1`, hence every node on layer t can be identified by a unique t-bit string `a`. We can assign a non-negative weight y_{a} to the node `a` and compute for every `a` its weighted-cost $y_{\mathsf{a}}\mathsf{cost}(\mathsf{a})$. Computing $g_n(v)$ for the linear program of Figure 3 is equivalent to finding a weight-assignment to every node in the tree (except the root node which corresponds to the empty string) that minimizes $\sum y_{\mathsf{a}}\mathsf{cost}(\mathsf{a})$ while satisfying the constraints that $\sum y_{\mathsf{a}} = v$ and the sum of weights along any path is less than or equal to 1 (Figure 5a). Computing $f_n(v)$ for the linear program of Figure 4 equates to finding a similar weighted-cost-minimizing weight-assignment, but over a slightly different n-layer tree, where a perfect binary sub-tree of $n-1$ layers connects to the root with a `0`-edge (Figure 5b). Claim 2 states that solving $g_n(v)$ equates to finding a u value to partition v into u and $v-u$ such that $f_n(u) + f_n(v-u) + (r-1)(v-u)$ is minimized (Figure 5c), where $f_n(u)$ is the minimum cost of the dash-line-delineated sub-tree with quote u and $f_n(v-u) + (r-1)(v-u)$ is the minimum cost of the rest (dot-line-delineated) part of the tree with quote $v-u$ (note the addend $(r-1)(v-u)$ is needed to compensate the fact that $f_n(v-u)$ assumes the weighted-cost contribution of the root-connecting `0`-edge is $(v-u)\cdot 1$ whereas here it is a `1`-edge that contributes a weighted-cost of $(v-u)\cdot r$). Claim 3 states that computing $f_{n+1}(v)$ can be viewed as finding the best t such that $v + t \cdot g_n((v-(1-t))/t)$ is minimized as $1-t$ weight is associated to the root of the n-layer sub-tree (Figure 5d). Note that g_n takes $(v-(1-t))/t$ as input because (a) after $(1-t)$ weight is assigned to the root of the sub-tree, only $v-(1-t)$ quote is left to the rest of the sub-tree; and (b) g_n requires all weights to be normalized to compare with 1, so we normalize $v-(1-t)$ to $(v-(1-t))/t$ and scales $g_n((v-(1-t))/t)$ back to $t \cdot g_n((v-(1-t))/t)$ afterwards. Finally, we stress that the addend v is needed to account for the weighted-cost contributed by the root-connecting `0`-edge.

Given the base case solutions g_1 and g_2, Claim 2 and Claim 3 suggest a recursive algorithm to compute g_n for every n: given g_k, we first derive f_{k+1} from g_k using Claim 3, then derive g_{k+1} from f_{k+1} using Claim 2. Also based on this observation, we prove by mathematical induction that g_n is a continuous piece-wise linear function expressible by a closed-form formula (Claim 4). Additionally, we show that the induction steps can be efficiently computed using *memoization* because there are only $O(n)$ non-zero entries in the coefficient arrays of g_n and f_n (Claim 6).

Claim 4. *Let $g_n(v)$ be the solution to the linear program of Figure 3. Then for every $n \in \mathbb{Z}^+$ and $v \in \mathbb{R}^+$, $v \leq 2^n$,*

$$g_n(v) = \sum_{i=0}^{2^n-1} G_n[i] \cdot \langle v - i \rangle,$$

where G_n is an array of 2^n non-negative constants uniquely determined by r (the cost ratio). Moreover, for every $n \geq 2$, $G_n[0] = 1, G_n[1] = r-1, G_n[2] = 2$, and $\forall\ 0 \leq i \leq 2^n - 1$, $G_n[i] \in \mathbb{Z}[r]$.



(a) Understanding $g_n(v)$

(b) Understanding $f_n(v)$

(c) Understanding Claim 2

(d) Understanding Claim 3

Figure 5: Intuitive interpretation of Claim 2 and Claim 3

Claim 5. *Let $f_n(v)$ be the solution to the linear program of Figure 4. Then for every $n \in \mathbb{Z}^+$ and $v \in \mathbb{R}^+$, $v \le 2^n$,*

$$f_n(v) = v + \sum_{i=0}^{2^{n-1}-1} F_n[i] \cdot \langle v - i \rangle,$$

where F_n is an array of 2^{n-1} non-negative constants uniquely determined by r (the cost ratio). Moreover, for every $n \ge 3$, $F_n[0] = 0, F_n[1] = r+1, F_n[2] = 1$, and $\forall\, 0 \le i \le 2^n - 1$, $F_n[i] \in \mathbb{Z}[r]$.

We prove the above two claims in Appendix C. In addition, the proof of the inductive step (Lemma 12) gives a simple algorithm to derive f_{n+1} from g_n: setting $F_{n+1}[0] \coloneqq 0$, $F_{n+1}[1] \coloneqq r+1$, $F_{n+1}[2] \coloneqq 1$ and $F_{n+1}[i] \coloneqq G_n[i]$ for all $i \ge 3$.

The proof of the inductive step (Lemma 13) also gives a simple two-step linear algorithm to derive G_n from F_n:

1. For $0 \le i \le 2^n - 1$, compute $A_n[i] \coloneqq 1 + \sum_{j=0}^{i} F_k[j]$ and $B_n[i] \coloneqq r + \sum_{j=0}^{i} F_k[j]$.
2. Merge A_n and B_n into a non-decreasing array H_n (with length $2^{n+1}-2$), and set $G_n[0] \coloneqq H_n[0]$ and $G_n[i] \coloneqq H_n[i] - H_n[i-1]$.

The key insight that establishes the efficiency of our approach is that G_n is fairly sparse, a fact we formally state below and prove in Appendix D.

Claim 6. *The array G_n contains at most $O(n)$ non-zero entries.*

Claim 6 implies the following corollary.

Corollary 7.

1. *The array F_n has at most $O(n)$ non-zero entries because F_n differs from G_{n-1} at only the first three entries.*
2. *For the array A_n defined in the proof of Lemma 13, there are at most $O(n)$ position i where $A_n[i] \ne A_n[i+1]$.*
3. *The piece-wise linear function $u_n(v)$ can be succinctly represented in $O(n)$ space, namely recording the $O(n)$ $A_n[i]$s positions' in H_n (where $A_n[i] \ne A_n[i+1]$).*

3.3 Obtaining An Optimal Strategy

Fixing n, we have shown how to efficiently compute the minimum expected price $\mathcal{C}$ has to pay to win the cut-and-choose game with probability at least $1-\varepsilon$. But what strategy should $\mathcal{C}$ use to actually achieve that minimum cost?

Note that a naïve representation of the randomized optimal strategy will require $O(2^n)$ variables. That is, an x_{a} for every $\mathsf{a} \in \{0,1\}^t, 0 \le t \le n$ is needed to denote $\mathcal{C}$'s strategy distribution. To address this exponential-space demand (in n), our key observation is that $\mathcal{C}$'s optimal strategy distribution can actually be represented in $O(n^2)$ space and it suffices to provide $\mathcal{C}$ an efficient algorithm to sample her cut-and-choose strategy from the optimal strategy distribution.

Optimal Strategy Sampling. Fixing r, we propose to represent $\mathcal{C}$'s optimal strategy by the family of functions $\{u_k \,|\, 1 \le k \le n\}$. Recall that Lemma 13 and Corollary 7 show that every u_k is a continuous piece-wise linear function describable in $O(n)$ space. Our sampling algorithm is described in Algorithm 1. The goal is to ensure every bit-string a being sampled with probability y^*_{a}/v where $\{y^*_{\mathsf{a}}\}_{\mathsf{a}\in\{0,1\}^n}$ solves the linear program of Figure 3. The correctness of our sampling algorithm is formalized in Claim 8 and proved in Appendix E. The proof also implies that the **while** loop (line 3 of Algorithm 1) runs at most n times.

Claim 8. *Let x^*_{a} be the probability of Algorithm 1 outputting string a, then $\{x^*_{\mathsf{a}}\}_{\mathsf{a}\in\{0,1\}^n}$ solves the linear program of Figure 2. Moreover, if $\varepsilon = 2^{-k}$ for some integer k, then $x^*_{\mathsf{a}} \in \{0, 2^{-k}\}$ for all*

Algorithm 1 $\mathsf{sample}(n, \varepsilon, r)$

Require: $\varepsilon \in \mathbb{R}^+$, $n \in \mathbb{Z}^+$ and $r > 1$.

Ensure: Any strategy string a is sampled with probability $y_{\mathsf{a}} \cdot \varepsilon = x_{\mathsf{a}}$.

1: Pre-compute $\{u_k \,|\, 1 \le k \le n\}$ from r as shown at the end of the proof of Lemma 13
2: $x \coloneqq 1/\varepsilon$; $k \coloneqq n$; $i \coloneqq 1$
3: **while** $x > 1$ **do**
4: $s \leftarrow [0, x)$ ▷ Uniformly sample a real number in $[0, x)$
5: **if** $1 < x \le 2$ **then**
6: **if** $s < 2 - x$ **then**
7: **return** a
8: **else if** $2 - x \le s < 1$ **then**
9: $\mathsf{a}_i \coloneqq 0$
10: **return** a
11: **else** ▷ $1 \le s < x$
12: $\mathsf{a}_i \coloneqq 1$
13: **return** a
14: **else** ▷ $x > 2$
15: **if** $s < u_k(x)$ **then**
16: $\mathsf{a}_i \coloneqq 0$
17: $x \coloneqq u_k(x)$
18: **else** ▷ $s \ge u_k(x)$
19: $\mathsf{a}_i \coloneqq 1$
20: $x \coloneqq x - u_k(x)$
21: $k \coloneqq k - 1$; $i \coloneqq i + 1$
22: **return** a

$\mathsf{a} \in \{0,1\}^n$.

Based on what we have shown so far, the proof of Theorem 1 is relatively straightforward.

Proof of Theorem 1. Because there are at most $O(n)$ non-zero entries in G_n, it suffices to only record the $O(n)$ non-zero entries of G_n and takes only $O(n)$ time to compute G_n from G_{n-1}. Thus, fixing n, our solver uses $O(n^2)$ time (to compute $G_1, \ldots, G_n$) and $O(n^2)$ space (as it memorizes $G_1, \ldots, G_n$ to enable efficient sampling by Algorithm 1). □

4 Solve $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}(r, \varepsilon)$ for unbounded t

We now know how to efficiently solve the linear program of Figure 2 for any fixed n, but it remains to solve the LP for $n \to \infty$. Note that the minimum cost of cut-and-choose cannot increase as n grows because an optimal strategy for $n = n_0$ is always a feasible strategy for $n = n_0 + 1$. Thus, an intuitive idea is to solve $\mathsf{ExtCnC}_{\mathcal{A},\mathcal{C}}$ for every possible integer n from $\lceil \log(1/\varepsilon) \rceil$ onward (note we ignore n values less than $\lceil \log(1/\varepsilon) \rceil$ because it is impossible for any such n to bound $\mathcal{A}$'s winning odds by ε). However, the soundness of this idea hinges on answers to the following two questions:

(1) Can we stop searching at some point without sacrificing optimality guarantee?

(2) How do we know when to stop?

Fortunately, we had positive answers to both: (1) a globally optimal strategy always use a finite n; and (2) we can stop searching once reaching some $\hat{n}$ such that $G_{\hat{n}}[i] = G_{\hat{n}+1}[i]$ for all $0 < i < \lceil 1/\varepsilon \rceil$. We formalize these results as Claim 10 and Claim 9, respectively, and prove them in Appendix F.

Claim 9. *If there exist $\hat{n}$ such that $G_{\hat{n}}[i] = G_{\hat{n}+1}[i]$ for all $0 \le i < 2^{\hat{n}}$, then $g_{\hat{n}}(v) = g_{\hat{n}+i}(v)$ for all positive integer i and real v $(0 < v \le 2^{\hat{n}})$.*

Claim 10. *For all ε $(0 < \varepsilon < 1)$, let $\hat{n}$ $(\hat{n} \ge \lceil \varepsilon \rceil)$ be the smallest integer such that $G_{\hat{n}}[i] = G_{\hat{n}+1}[i]$ for all $0 \le i < 2^{\hat{n}}$, and $g_\infty(1/\varepsilon)$ be the minimal cost of the optimal strategy that allows infinite-length challenges, then $g_{\hat{n}}(1/\varepsilon) \le g_\infty(1/\varepsilon)$.*

5 Results and Applications

We have implemented the proposed solver and experimentally evaluated our approach. Our solver is open-sourced at `https://github.com/Opt-Cut-N-Choose`. In comparison with the best existing solutions [23] which assumed fixed-length challenges, our solution is more than **2x** more effective in reducing the adversary's winning odds (Figure 6).

Let $\hat{n}$ be the upper-bound on the length of the challenges in an optimal strategy. Note that $\hat{n}$ always exists as per Claim 10 but its value varies with r and ε. Figure 7 plots $\hat{n}$ for a number of (r, ε) combinations calculated by our search algorithm described in Section 4. We observe that, for a fixed ε, $\hat{n}$ grows roughly linearly with r, while a smaller ε implies a steeper slope.

Apply to Secure Computation. We propose a garbled-circuit-based actively-secure computation protocol leveraging the above analysis of the extended cut-and-choose game. The full protocol and its security proof are described in Appendix G. We stress that allowing the length of the challenge string to vary does not incur extra communication rounds: simply allowing the circuit generator (the $\mathcal{A}$) to receive the "stop" signal from the circuit evaluator (the $\mathcal{C}$) *asynchronously*. I.e., the generator only needs to poll the asynchronous communication channel for a "stop" signal but never needs to stop sending circuits (using a different communication channel) during the process.

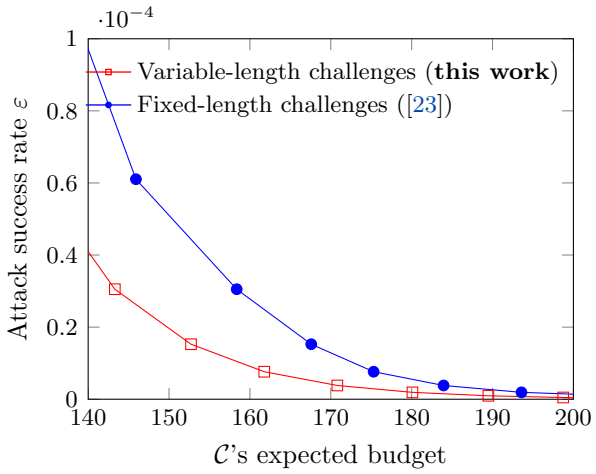


Figure 6: Better thwarting attacks.

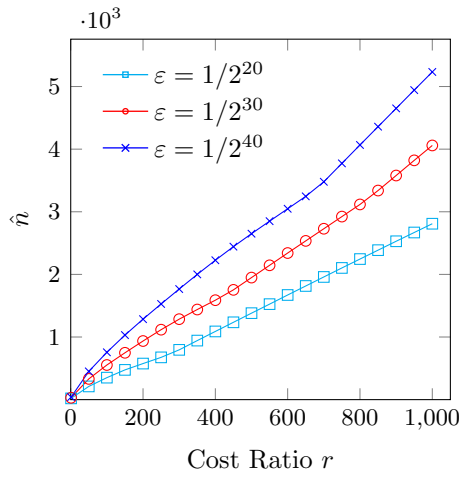


Figure 7: Bounds of n in optimal strategies

Apply to Prefix-free Encoding of Encryptions. It is relatively straightforward to derive the encode and decode functions based on the sample algorithm in Algorithm 1. The high-level idea is, for encoding, we assign the input symbol to s (instead of a uniformly sampled s) to drive sample; for decoding, we parse the codeword a backwards to determine the original code w. We formally present encode (Algorithm 2) and decode (Algorithm 3) in Appendix H. Compared to normal ways of storing/transmitting encryptions, by exploiting the 0/1 cost gaps, our approach can save 17% (when $r = 5$) to 29% (when $r = 10$). In general, our savings increase when longer input ciphertext-symbols are considered.

6 Conclusion

We revisited a cost-aware mathematical modeling of cut-and-choose games and quantitatively evaluate a challenger's benefits to employ variable-length challenges. Although the resulting mathematical program involves infinitely many variables and constraints, it can be efficiently solved with our quadratic solver. The techniques used in our work would be of independent interest to other optimization problems exhibiting similar structures.

References

[1] A. Afshar, P. Mohassel, B. Pinkas, and B. Riva. Non-interactive secure computation based on cut-and-choose. In *EUROCRYPT*, 2014.

[2] S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to better - how to make Bitcoin a better currency. In *International Conference on Financial Cryptography and Data Security*, 2012.

[3] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, 2012.

[4] Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Sorting with asymmetric read and write costs. In *ACM Symposium on Parallelism in Algorithms and Architectures*, 2015.

[5] M. Blum. How to prove a theorem so no one else can claim it, August 1986. International Congress of Mathematicians.

[6] Phil Bradford, Mordecai J Golin, Lawrence L Larmore, and Wojciech Rytter. Optimal prefix-free codes for unequal letter costs: Dynamic programming with the monge property. *Journal of Algorithms*, 42(2):277–303, 2002.

[7] L. Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique. In *ASIACRYPT*, 2013.

[8] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[9] K.-M. Chung, Y. Kalai, and S. P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, 2010.

[10] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In *CRYPTO*, 2004.

[11] Travis Gagie. Dynamic shannon coding. *Information Processing Letters*, 102(2-3):113–117, 2007.

[12] O. Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[13] Mordecai Golin and Jian Li. More efficient algorithms and analyses for unequal letter cost prefix-free coding. *IEEE Transactions on Information Theory*, 54(8):3412–3424, 2008.

[14] Mordecai J Golin, Claire Kenyon, and Neal E Young. Huffman coding with unequal letter costs. In *ACM Symposium on Theory of Computing*, pages 785–791. ACM, 2002.

[15] Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO*, 2013.

[16] Alon Itai. Optimal alphabetic trees. *SIAM Journal on Computing*, 5(1):9–18, 1976.

[17] Yongsoo Joo, Youngjin Cho, Donghwa Shin, Jaehyun Park, and Naehyuck Chang. An energy characterization platform for memory devices and energy-aware data compression for multilevel-cell flash memory. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 13(3):43, 2008.

[18] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, 2013.

[19] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.

[20] Moinuddin K. Qureshi, Michele M. Franceschini, Ashish Jagmohan, and Luis A. Lastras. Preset: Improving performance of phase change memories by exploiting asymmetry in write times. In *International Symposium on Computer Architecture*, 2012.

[21] X. Wang. The emp-toolkit. https://github.com/emp-toolkit/.

[22] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.

[23] R. Zhu, Y. Huang, A. Shelat, and J. Katz. The cut-and-choose game and its application to cryptographic protocols. In *USENIX Security Symposium*, 2016.

A Finding $g_1(v), g_2(v)$

Finding $g_1(v)$. When $n = 1$, we know $\mathsf{a} \in \{0, 1\}$, $\mathsf{cost}(0) = \mathsf{zeros}(0) + r \cdot \mathsf{ones}(0) = 1, \mathsf{cost}(1) = \mathsf{zeros}(1) + r \cdot \mathsf{ones}(1) = r$. Thus, the target linear program reduces to

$$\min \quad y_0 \cdot 1 + y_1 \cdot r$$

subject to

$$\begin{aligned} y_0, y_1 &\geq 0 \\ y_0 + y_1 &= v \\ y_0 &\leq 1 \\ y_1 &\leq 1. \end{aligned}$$

This linear program can be solved analytically based on three observations:

(1) Because $y_1 = v - y_0$, then $y_0 + y_1 r = y_0 + (v - y_0)r = vr + (1 - r)y_0$. Also because $v > 0, r \geq 1$, we know $y_0 + y_1 r$ is minimized when y_0 is maximized.

(2) $y_0 = v - y_1$ and $y_1 \geq 0$ imply that $y_0 \leq v$. Also because the constraints have that $y_0 \leq 1$, we know the maximal value of y_0 hinges on whether v is smaller than 1.

(3) $y_0 \leq 1, y_1 \leq 1 \Rightarrow v = y_0 + y_1 \leq 2$.

Therefore, the solution with respect to $n = 1$ is:

1. If $0 \leq v < 1$, $y_0 = v, y_1 = 0$ minimizes $y_0 + y_1 r$ to v. Namely, $\forall v \in (0, 1], g_1(v) = v$.
2. If $1 \leq v \leq 2$, setting $y_0 = 1$ and $y_1 = v - 1$ minimizes $y_0 + y_1 r$ to $1 + r(v - 1)$. Namely, $\forall v \in (1, 2], g_1(v) = 1 + r(v - 1)$.

Simply put,

$$g_1(v) = \begin{cases} v & 0 \leq v \leq 1 \\ v + (r - 1)(v - 1) & 1 < v \leq 2. \end{cases} \qquad \square$$

Finding $g_2(v)$. When $n = 2$, we know $\mathsf{a} \in \{0, 1, 00, 01, 10, 11\}$. $\mathsf{cost}(00) = 2, \mathsf{cost}(01) = \mathsf{cost}(10) = r + 1, \mathsf{cost}(11) = 2r$, hence the problem reduces to

$$\min \quad y_0 \cdot 1 + y_1 \cdot r + y_{00} \cdot 2 + (y_{01} + y_{10}) \cdot (r + 1) + y_{11} \cdot 2r$$

subject to

$$\begin{aligned} y_0, y_1, y_{00}, y_{01}, y_{10}, y_{11} &\geq 0 \\ y_0 + y_1 + y_{00} + y_{01} + y_{10} + y_{11} &= v \\ y_0 + y_{00} &\leq 1 \\ y_0 + y_{01} &\leq 1 \\ y_1 + y_{10} &\leq 1 \\ y_1 + y_{11} &\leq 1. \end{aligned}$$

Note that

$$\begin{aligned}\mathsf{target} &= y_{\mathsf{0}} \cdot 1 + y_{\mathsf{1}} \cdot r + y_{\mathsf{00}} \cdot 2 + (y_{\mathsf{01}} + y_{\mathsf{10}}) \cdot (r+1) + y_{\mathsf{11}} \cdot 2r \\ &\geq (y_{\mathsf{0}} + y_{\mathsf{00}}) \cdot 1 + (y_{\mathsf{1}} + y_{\mathsf{01}} + y_{\mathsf{10}} + y_{\mathsf{11}}) \cdot r \\ &= (y_{\mathsf{0}} + y_{\mathsf{00}}) \cdot 1 + [v - (y_{\mathsf{0}} + y_{\mathsf{00}})] \cdot r \\ &= v \cdot r + (y_{\mathsf{0}} + y_{\mathsf{00}}) \cdot (1-r).\end{aligned}$$

Because $y_{\mathsf{0}} + y_{\mathsf{00}} \leq 1$ and $y_{\mathsf{0}} + y_{\mathsf{00}} \leq \sum_{\mathsf{len(a)} \leq 2} y_{\mathsf{a}} = v$, so $\mathsf{target} \geq v$ if $0 \leq v \leq 1$ and $\mathsf{target} \geq vr + (1-r) = 1 + r(v-1) = v + (r-1)(v-1)$ if $1 < v \leq 2$. Indeed,

1. If $v \in [0,1]$, setting $y_{\mathsf{0}} = v$ and $y_{\mathsf{1}} = y_{\mathsf{00}} = y_{\mathsf{01}} = y_{\mathsf{10}} = y_{\mathsf{11}} = 0$ allows target to achieve its lower bound v.

2. If $v \in (1,2]$, setting $y_{\mathsf{0}} = 1$, $y_{\mathsf{1}} = v - 1$, and $y_{\mathsf{00}} = y_{\mathsf{01}} = y_{\mathsf{10}} = y_{\mathsf{11}} = 0$ allows target to achieve its lower bound $v + (r-1)(v-1)$.

Therefore, if $0 \leq v \leq 1$, then $g_2(v) = v$; and if $1 < v \leq 2$, then $g_2(v) = v + (r-1)(v-1)$.

Now consider the case when $v > 2$. Let $v_1 = y_{\mathsf{0}} + y_{\mathsf{00}} + y_{\mathsf{01}}$ and $v_2 = y_{\mathsf{1}} + y_{\mathsf{10}} + y_{\mathsf{11}}$, so the equality constraint translate to $v_1 + v_2 = v$. Note that when target is minimized, we must have $v_1 \geq v_2$ (otherwise, if $v_1 < v_2$, let $t = \min \mathsf{target}$, achieved with a particular assignment of $y_{\mathsf{0}}, y_{\mathsf{1}}, y_{\mathsf{00}}, y_{\mathsf{01}}, y_{\mathsf{10}}, y_{\mathsf{11}}$, we can always swap the values of y_{0} (y_{00}, y_{01}, respectively) and y_{1} (y_{10}, y_{11}, respectively) to obtain $t' = \mathsf{target}$ such that $t' - t = y_{\mathsf{0}}(r-1) + y_{\mathsf{1}}(1-r) + y_{\mathsf{00}}(r-1) + y_{\mathsf{01}}(r-1) + y_{\mathsf{10}}(1-r) + y_{\mathsf{11}}(1-r) = (v_1 - v_2)(r-1) < 0$, which contradicts with the fact that $t = \min \mathsf{target}$). Moreover, $v_1 = y_{\mathsf{0}} + y_{\mathsf{00}} + y_{\mathsf{01}} \leq (y_{\mathsf{0}} + y_{\mathsf{00}}) + (y_{\mathsf{0}} + y_{\mathsf{01}}) \leq 2$. Similarly $v_2 \leq 2$.

Because $y_{\mathsf{00}} = v_1 - (y_{\mathsf{0}} + y_{\mathsf{01}}) \geq 0$ and $y_{\mathsf{00}} = v_1 - (y_{\mathsf{0}} + y_{\mathsf{01}}) \geq v_1 - 1$, so $y_{\mathsf{00}} \geq \max\{v_1 - 1, 0\}$. Similarly, $y_{\mathsf{01}} \geq \max\{v_1 - 1, 0\}$. Thus,

$$\begin{aligned}y_{\mathsf{0}} \cdot 1 + y_{\mathsf{00}} \cdot 2 + y_{\mathsf{01}} \cdot (1+r) &= (y_{\mathsf{0}} + y_{\mathsf{00}} + y_{\mathsf{01}}) + y_{\mathsf{00}} + y_{\mathsf{01}} \cdot r \\ &\geq v_1 + \max\{v_1 - 1, 0\} + \max\{v_1 - 1, 0\} \cdot r \\ &= v_1 + \max\{v_1 - 1, 0\} \cdot (1+r).\end{aligned}$$

Similarly, we can derive

$$\begin{aligned}y_{\mathsf{1}} \cdot r + y_{\mathsf{11}} \cdot 2r + y_{\mathsf{10}} \cdot (1+r) &= (y_{\mathsf{1}} + y_{\mathsf{10}} + y_{\mathsf{11}}) \cdot r + y_{\mathsf{10}} + y_{\mathsf{11}} \cdot r \\ &\geq v_2 \cdot r + \max\{v_2 - 1, 0\} + \max\{v_2 - 1, 0\} \cdot r \\ &= v_2 \cdot r + \max\{v_2 - 1, 0\} \cdot (1+r).\end{aligned}$$

Thus,

$$\begin{aligned}\mathsf{target} &= y_{\mathsf{0}} \cdot 1 + y_{\mathsf{00}} \cdot 2 + y_{\mathsf{01}} \cdot (1+r) + y_{\mathsf{1}} \cdot r + y_{\mathsf{11}} \cdot 2r + y_{\mathsf{10}} \cdot (1+r) \\ &= v_1 + \max\{v_1 - 1, 0\} \cdot (1+r) + v_2 \cdot r + \max\{v_2 - 1, 0\} \cdot (1+r) \\ &= v_1 + v_2 \cdot r + (\max\{v_1 - 1, 0\} + \max\{v_2 - 1, 0\})(1+r)\end{aligned}$$

Because $v = v_1 + v_2 > 2$, it has to be the case that either $v_1 \geq 1$ or $v_2 \geq 1$.

1. If $v_2 \leq 1$, then $v_1 = v - v_2 \geq v - 1 \geq 2 - 1 = 1$. Hence,

$$\begin{aligned}\mathsf{target} &\geq v_1 + v_2 \cdot r + (\max\{v_1 - 1, 0\} + \max\{v_2 - 1, 0\})(1+r) \\ &= v_1 + v_2 \cdot r + (v_1 - 1)(1+r) \\ &= v_1 + (v - v_1) \cdot r + (v_1 - 1)(1+r) \\ &= 2v_1 + vr - (1+r) \\ &\geq 2(v-1) + vr - (1+r) \\ &= (2+r) \cdot v - r - 3.\end{aligned}$$

2. If $v_2 \geq 1$, we know $v_1 \geq v_2 \geq 1$ and $v_1 = v - v_2 \leq v - 1$. Hence,

$$\begin{aligned}
\mathsf{target} &\geq v_1 + v_2 \cdot r + (\max\{v_1 - 1, 0\} + \max\{v_2 - 1, 0\})(1 + r) \\
&= v_1 + v_2 \cdot r + (v_1 - 1 + v_2 - 1)(1 + r) \\
&= v_1 + v_2 \cdot r + (v - 2)(1 + r) \\
&= v_1 + (v - v_1) \cdot r + (v - 2)(1 + r) \\
&= v \cdot r + v_1 \cdot (1 - r) + (v - 2) \cdot (1 + r) \\
&\geq v \cdot r + (v - 1) \cdot (1 - r) + (v - 2) \cdot (1 + r) \\
&= (2 + r) \cdot v - r - 3.
\end{aligned}$$

When $v \in [2, 3]$, we can verify that setting $y_{\mathsf{0}} = 3 - v, y_{\mathsf{00}} = y_{\mathsf{01}} = v - 2, y_{\mathsf{1}} = 1, y_{\mathsf{10}} = y_{\mathsf{11}} = 0$ actually achieves this lower bound, i.e., $\mathsf{target} = (2 + r) \cdot v - r - 3$.

Finally, when $v \in [3, 4]$, we must have $v_1 \geq 1$ and $v_2 \geq 1$. Thus,

$$\begin{aligned}
\mathsf{target} &\geq v_1 + v_2 \cdot r + (\max\{v_1 - 1, 0\} + \max\{v_2 - 1, 0\})(1 + r) \\
&= v_1 + v_2 \cdot r + (v_1 - 1 + v_2 - 1)(1 + r) \\
&= v_1 + v_2 \cdot r + (v - 2)(1 + r) \\
&= v_1 + (v - v_1) \cdot r + (v - 2)(1 + r) \\
&= v \cdot r + v_1 \cdot (1 - r) + (v - 2) \cdot (1 + r) \\
&\geq v \cdot r + 2 \cdot (1 - r) + (v - 2) \cdot (1 + r) \\
&= v \cdot (2r + 1) - 4r.
\end{aligned}$$

In this case, we can verify that setting $y_{\mathsf{0}} = 0, y_{\mathsf{00}} = y_{\mathsf{01}} = 1, y_{\mathsf{1}} = 4 - v, y_{\mathsf{10}} = y_{\mathsf{11}} = v - 3$ actually achieves this lower bound of target.

In summary,

$$g_2(v) = \begin{cases} v, & 0 \leq v \leq 1 \\ v + (r - 1)(v - 1), & 1 < v \leq 2 \\ (2 + r) \cdot v - r - 3 = v + (r - 1)(v - 1) + 2(v - 2), & 2 < v \leq 3 \\ (2r + 1) \cdot v - 4r = v + (r - 1)(v - 1) + 2(v - 2) + (r - 1)(v - 3), & 3 < v \leq 4. \end{cases}$$ □

B Proofs of Claim 2 and Claim 3

Claim 2. *For any $v \in \mathbb{R}^+$ and $v \leq 2^n$,*

$$g_n(v) = \min_{0 \leq u \leq v} \{\, f_n(u) + f_n(v - u) + (r - 1)(v - u) \,\}.$$

For every r, we thus define $u_n(v)$ to be the u value that minimizes $g_n(v)$.

Proof.

$$
\begin{aligned}
g_n(v) &= \min_{(1)(2)(5)(6)} \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \\
&= \min_{(1)(2)(5)(6)} \left\{ \sum_{\mathsf{a}=0\|\{0,1\}^*} y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) + \sum_{\mathsf{a}=1\|\{0,1\}^*} y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\} \\
&= \min_{0\le u\le v} \left\{ \min_{\substack{(1)(2)(6) \\ \sum y_{\mathsf{a}}=u \\ \mathsf{a}=0\|\{0,1\}^*}} \left\{ \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\} + \min_{\substack{(1)(2)(6) \\ \sum y_{\mathsf{a}}=v-u \\ \mathsf{a}=1\|\{0,1\}^*}} \left\{ \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\} \right\} \\
&= \min_{0\le u\le v} \left\{ \min_{\substack{(1)(2)(6)(7) \\ \sum y_{\mathsf{a}}=u}} \left\{ \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\} + \min_{\substack{(1)(2)(6) \\ \sum y_{\mathsf{a}}=v-u \\ \mathsf{a}=1\|\{0,1\}^*}} \left\{ \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\} \right\} \\
&= \min_{0\le u\le v} \left\{ f_n(u) + \min_{\substack{(1)(2)(6) \\ \sum y_{\mathsf{a}}=v-u \\ \mathsf{a}=1\|\{0,1\}^*}} \left\{ \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\} \right\} \\
&= \min_{0\le u\le v} \left\{ f_n(u) + \min_{\substack{(1)(2)(6) \\ \sum y_{\mathsf{a}}=v-u \\ \mathsf{a}=0\|\{0,1\}^*}} \left\{ \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\} + (r-1)(v-u) \right\} \\
&= \min_{0\le u\le v} \left\{ f_n(u) + f_n(v-u) + (r-1)(v-u) \right\}.
\end{aligned}
$$

□

Claim 3. *For any $v \in \mathbb{R}^+$, $v \le 2^n$ and integer $n \ge 1$,*

$$
f_{n+1}(v) = v + \min_{\max\left(0,\, \frac{v-1}{2^n-1}\right) \le t \le 1} \left\{ t \cdot g_n\big((v+t-1)/t \big) \right\}.
$$

Proof. Let $y_0 = 1 - t$ where $0 \le t \le 1$. Then

$$
\begin{aligned}
f_{n+1}(v) &= \min_{(1)(2)(5)(6)(7)} \left\{ \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\} \\
&= \min_{0\le t\le 1} \left\{ \min_{\substack{(1)(2)(5)(6)(7) \\ y_0=1-t}} \left\{ \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\} \right\}
\end{aligned}
$$

Expand $\min_{\substack{(1)(2)(5)(6)(7) \\ y_0=1-t}} \left\{ \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \right\}$ into its LP form, we get

$$\begin{aligned}
&\min \quad \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \\
&\text{subject to} \\
&\qquad y_{\mathsf{0}} = 1 - t \\
&\qquad y_{\mathsf{a}} = 0, \quad \forall\, \mathsf{a} = \mathsf{1} \| \{\mathsf{0},\mathsf{1}\}^t, 0 \le t \le n \\
&\qquad \mathsf{len}(\mathsf{a}) \le n+1 \\
&\qquad y_{\mathsf{a}} \ge 0, \quad \forall\, \mathsf{a} \in \{\mathsf{0},\mathsf{1}\}^t, 1 \le t \le n+1 \\
&\qquad \sum y_{\mathsf{a}} = v \\
&y_{\mathsf{a}_1} + y_{\mathsf{a}_1\mathsf{a}_2} + \cdots + y_{\mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_{n+1}} \le 1, \quad \forall\, \mathsf{a} = \mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_{n+1} \in \{\mathsf{0},\mathsf{1}\}^{n+1}.
\end{aligned}$$

That is,

$$\begin{aligned}
&\min \quad \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \\
&\text{subject to} \\
&\qquad y_{\mathsf{0}} = 1 - t \\
&\qquad y_{\mathsf{a}} = 0, \quad \forall\, \mathsf{a} = \mathsf{1} \| \{\mathsf{0},\mathsf{1}\}^t, 0 \le t \le n \\
&\qquad \mathsf{len}(\mathsf{a}) \le n+1 \\
&\qquad y_{\mathsf{a}} \ge 0, \quad \forall\, \mathsf{a} \in \{\mathsf{0},\mathsf{1}\}^t, 1 \le t \le n+1 \\
&\qquad y_{\mathsf{0}} + \sum_{\mathsf{a} \ne \mathsf{0}} y_{\mathsf{a}} = v \\
&y_{\mathsf{0}} + y_{\mathsf{0}\mathsf{a}_2} + \cdots + y_{\mathsf{0}\mathsf{a}_2\ldots\mathsf{a}_{n+1}} \le 1, \quad \forall\, \mathsf{a} = \mathsf{0}\,\mathsf{a}_2\ldots\mathsf{a}_{n+1} \in \{\mathsf{0},\mathsf{1}\}^{n+1}.
\end{aligned}$$

Fixing t, the linear program can be simplified to

$$\begin{aligned}
&v + \min \quad \sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \\
&\text{subject to} \\
&\qquad \mathsf{len}(\mathsf{a}) \le n \\
&\qquad y_{\mathsf{a}} \ge 0, \quad \forall\, \mathsf{a} \in \{\mathsf{0},\mathsf{1}\}^t, 1 \le t \le n \\
&\qquad \sum y_{\mathsf{a}} = v + t - 1 \\
&y_{\mathsf{a}_1} + y_{\mathsf{a}_1\mathsf{a}_2} + \cdots + y_{\mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n} \le t, \quad \forall\, \mathsf{a} = \mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n \in \{\mathsf{0},\mathsf{1}\}^n.
\end{aligned}$$

Let $y'_{\mathsf{a}} = y_{\mathsf{a}}/t$, we can reformulate the linear program as

$$\begin{aligned}
&v + t \cdot \min \quad \sum y'_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a}) \\
&\text{subject to} \\
&\qquad \mathsf{len}(\mathsf{a}) \le n \\
&\qquad y'_{\mathsf{a}} \ge 0, \quad \forall\, \mathsf{a} \in \{\mathsf{0},\mathsf{1}\}^t, 1 \le t \le n \\
&\qquad \sum y'_{\mathsf{a}} = (v + t - 1)/t \\
&y'_{\mathsf{a}_1} + y'_{\mathsf{a}_1\mathsf{a}_2} + \cdots + y'_{\mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n} \le 1, \quad \forall\, \mathsf{a} = \mathsf{a}_1\mathsf{a}_2\ldots\mathsf{a}_n \in \{\mathsf{0},\mathsf{1}\}^n.
\end{aligned}$$

which is exactly $v + \min_{0\le t\le 1}\{t \cdot g_n((v+t-1)/t))\}$. Thus,

$$f_{n+1}(v) = \min_{0\le t\le 1}\left\{ \min_{\substack{(1)(2)(5)(6)(7)\\ y_0=1-t}} \left\{\sum y_{\mathsf{a}} \cdot \mathsf{cost}(\mathsf{a})\right\}\right\}$$
$$= v + \min_{0\le t\le 1}\Big\{\, t \cdot g_n\big(\,(v+t-1)/t\,\big)\,\Big\}.$$

Finally, because $(v+t-1)/t = \sum y'_{\mathsf{a}} \le 2^n$ and $t \ge 0$, we additionally know $t \ge \max\big(0,\ \frac{v-1}{2^n-1}\big)$. □

C Proofs of Claim 4 and Claim 5

Claim 4. *Let $g_n(v)$ be the solution to the linear program of Figure 3. Then for every $n \in \mathbb{Z}^+$ and $v \in \mathbb{R}^+$, $v \le 2^n$,*

$$g_n(v) = \sum_{i=0}^{2^n-1} G_n[i] \cdot \langle v - i\rangle,$$

where G_n is an array of 2^n non-negative constants uniquely determined by r (the cost ratio). Moreover, for every $n \ge 2$, $G_n[0] = 1, G_n[1] = r-1, G_n[2] = 2$, and $\forall\ 0 \le i \le 2^n - 1$, $G_n[i] \in \mathbb{Z}[r]$.

Proof. This can be proved using mathematical induction: (1) Lemma 11 shows the base case; (2) Given that Claim 4 holds for $n = k$, applying Lemma 12 and Lemma 13, we know Claim 4 must hold for $n = k+1$. Hence completing the proof. □

Claim 5. *Let $f_n(v)$ be the solution to the linear program of Figure 4. Then for every $n \in \mathbb{Z}^+$ and $v \in \mathbb{R}^+$, $v \le 2^n$,*

$$f_n(v) = v + \sum_{i=0}^{2^{n-1}-1} F_n[i] \cdot \langle v - i\rangle,$$

where F_n is an array of 2^{n-1} non-negative constants uniquely determined by r (the cost ratio). Moreover, for every $n \ge 3$, $F_n[0] = 0, F_n[1] = r+1, F_n[2] = 1$, and $\forall\ 0 \le i \le 2^n - 1$, $F_n[i] \in \mathbb{Z}[r]$.

Proof. This is a corollary from direct application of Claim 4 and Lemma 12. □

Lemma 11. *Claim 4 holds for $n = 1$ and $n = 2$.*

Proof. We have shown in Section 3.1 that $g_1(v) = v$ for $0 \le v \le 1$ and $g_1(v) = v + (r-1)(v-1)$ for $1 < v \le 2$. Thus Claim 4 holds at $n = 1$, with $G_n[0] = 1$, $G_n[1] = r - 1$. Similarly, the analytical form of $g_2(v)$ shown in Section 3.1 implies that Claim 4 holds for $n = 2$. □

Lemma 12. *If Claim 4 holds for $n \le k$, then Claim 5 holds for $n = k+1$.*

Proof. By Claim 3, we know that $f_{k+1}(v) = v + \min_{\max\left(0,\ \frac{v-1}{2^k-1}\right)\le t\le 1} t \cdot g_k\left(\dfrac{v+t-1}{t}\right)$. In addition, if

Claim 4 holds for $n \le k$, i.e., $g_k(v) = \langle v \rangle + (r-1)\langle v-1 \rangle + 2\langle v-2 \rangle + \sum_{i=3}^{2^k-1} G_k[i] \cdot \langle v-i \rangle$, then

$$
\begin{aligned}
f_{k+1}(v) &= v + \min_{\max\left(0, \frac{v-1}{2^k-1}\right) \le t \le 1} t \cdot g_k\left(\frac{v+t-1}{t}\right) \\
&= v + \min_{\max\left(0, \frac{v-1}{2^k-1}\right) \le t \le 1} t \cdot \left(\left\langle \frac{v+t-1}{t} \right\rangle + (r-1)\left\langle \frac{v+t-1}{t} - 1 \right\rangle + 2\left\langle \frac{v+t-1}{t} - 2 \right\rangle \right. \\
&\qquad \left. + \sum_{i=3}^{2^k-1} G_k[i] \cdot \left\langle \frac{v+t-1}{t} - i \right\rangle \right) \\
&= v + \min_{\max\left(0, \frac{v-1}{2^k-1}\right) \le t \le 1} \left(\langle v+t-1 \rangle + (r-1)\langle v+t-1-t \rangle + 2\langle v+t-1-2t \rangle \right. \\
&\qquad \left. + \sum_{i=3}^{2^k-1} G_k[i] \cdot \langle v+t-1-i \cdot t \rangle \right) \\
&= v + \min_{\max\left(0, \frac{v-1}{2^k-1}\right) \le t \le 1} \left(\langle v+t-1 \rangle + (r-1)\langle v-1 \rangle + 2\langle v-t-1 \rangle \right. \\
&\qquad \left. + \sum_{i=3}^{2^k-1} G_k[i] \cdot \langle v-1-(i-1) \cdot t \rangle \right)
\end{aligned}
$$

Let $M(t) = \langle v+t-1 \rangle + (r-1)\langle v-1 \rangle + 2\langle v-t-1 \rangle + \sum_{i=3}^{2^k-1} G_k[i] \cdot \langle v-1-(i-1) \cdot t \rangle$. Thus $f_{k+1}(v) = v + \min_{\max\left(0, \frac{v-1}{2^k-1}\right) \le t \le 1} M(t)$. Next, we consider how to select t to minimize $M(t)$.

1. When $0 \le v < 1$, because $\langle v-1 \rangle = 0$, $\langle v-1-t \rangle = 0$, $\langle v-1-(i-1) \cdot t \rangle = 0$, hence $M(t)$ reduces to $\langle v+t-1 \rangle$, which can be minimized to 0 when $t \le 1-v$.

2. When $1 \le v < 2$, $M(t)$ is always minimized when $t = v-1$ because

   (a) If $t \ge v-1$, $M(t)$ reduces to $v+t-1+(r-1)(v-1)$, which always increases with t, hence can be minimized when $t = v-1$ (note $v < t+1$, so $t \ge v-1$).

   (b) If $t \le v-1$,

   $$
   \begin{aligned}
   M(t) &= v+t-1+(r-1)(v-1)+2(v-t-1)+\sum_{i=3}^{\lfloor \frac{v-1}{t} \rfloor + 1} G_k[i] \cdot (v-1-(i-1) \cdot t) \\
   &= r \cdot (v-1) + 2 \cdot (v-1) - t + \sum_{i=3}^{\lfloor \frac{v-1}{t} \rfloor + 1} G_k[i] \cdot (v-1-(i-1) \cdot t)
   \end{aligned}
   $$

   which is a decreasing function of t, thus is minimized also when $t = v-1$ (note $1+t \le v$, so $t \le v-1$).

Therefore,

$$\begin{aligned} f_{k+1}(v) &= v + \min_{\max(0, \frac{v-1}{2^k-1}) \le t \le 1} t \cdot g_k((v+t-1)/t) \\ &= v + (v-1) \cdot g_k(2) \\ &= v + (v-1) \cdot (G_k[0] \cdot 2 + G_k[1] \cdot (2-1)) \\ &= v + (r+1)(v-1) \end{aligned}$$

3. When $v \ge 2$, (like the case when $t \le v-1$),

$$M(t) = r \cdot (v-1) + 2 \cdot (v-1) - t + \sum_{i=3}^{\lfloor \frac{v-1}{t} \rfloor + 1} G_k[i] \cdot (v - 1 - (i-1) \cdot t),$$

is also decreasing with t. However, this time, $M(t)$ is minimized when $t = 1$, because the only bound that t is subject to is $(v-1)/(2^k-1) \le t \le 1$. Thus, $f_{k+1}(v) = v + g_k(v)$.

In summary, we have shown that

$$f_{k+1}(v) = \begin{cases} v & 0 \le v < 1 \\ v + (r+1)(v-1) & 1 \le v < 2 \\ v + g_k(v) & 2 \le v \end{cases}$$

which can be equivalently re-written as

$$\begin{aligned} f_{k+1}(v) &= v + (r+1)\langle v-1 \rangle + \Big(g_k(v) - v - (r-1)\langle v-1 \rangle - \langle v-2 \rangle \Big) \\ &= v + (r+1)\langle v-1 \rangle + \langle v-2 \rangle + \sum_{i=3}^{2^k-1} G_k[i] \langle v-i \rangle \end{aligned}$$

Thus, Claim 5 holds for $n = k+1$. □

Lemma 13. *If Claim 5 holds for $n \le k$, then Claim 4 holds for $n = k$.*

Proof. By Claim 2, $g_k(v) = \min_{0 \le u \le v} \Big(f_k(u) + f_k(v-u) + (r-1)(v-u) \Big)$. Assuming Claim 5 holds, i.e., $f_k(v) = v + \sum_{i=0}^{2^k-1} F_k[i] \cdot \langle v-i \rangle$, our goal is to show that there exist an array $G_k = [1, r-1, 2, \dots]$ such that $g_k(v) = \sum_{i=0}^{2^k-1} G_k[i] \cdot \langle v-i \rangle$.

Let $\widetilde{f}_k(x) = f_k(x) + (r-1)x$, so $g_k(v) = \min_{0 \le u \le v} \Big(f_k(u) + \widetilde{f}_k(v-u) \Big)$. We also know that

$$\begin{aligned} \widetilde{f}_k(v) &= f_k(v) + (r-1) \cdot v \\ &= v + \sum_{i=0}^{2^k-1} F_k[i] \cdot \langle v-i \rangle + (r-1) \cdot v \\ &= r \cdot v + \sum_{i=0}^{2^k-1} F_k[i] \cdot \langle v-i \rangle. \end{aligned}$$

We further calculate the derivatives of $f_k(v)$ and $\widetilde{f}_k(v)$ as follows,

$$f_k'(v) = \left(v + \sum_{i=0}^{2^k-1} F_k[i]\langle v-i\rangle\right)' = 1 + \sum_{i=0}^{\lfloor v\rfloor} F_k[i]$$
$$\widetilde{f}_k{}'(v) = \left(vr + \sum_{i=0}^{2^k-1} F_k[i]\langle v-i\rangle\right)' = r + \sum_{i=0}^{\lfloor v\rfloor} F_k[i]$$

Solving $\min\limits_{0\le u\le v}\left(f_k(u) + \widetilde{f}_k(v-u)\right)$ can be best illustrated with an analogy. For a fixed v, this minimization problem is equivalent to the following *packing* problem: given two arrays (denoted by A_k and B_k, respectively) of unit-volume objects, whose weights are $A_k[i] = 1 + \sum_{j=0}^{i} F_k[j]$ and $B_k[i] = r + \sum_{j=0}^{i} F_k[j]$ ($1 \le i \le v$, note that A and B are non-decreasing arrays because every entry in F_k is non-negative), find the minimum weight of a pile of volume-v objects picked from the two arrays (partial picking is allowed). This is because

$$f_k(u) = \sum_{i=0}^{\lfloor u\rfloor-1} A_k[i] + (u - \lfloor u\rfloor)\cdot A_k[\lfloor u\rfloor]$$
$$\widetilde{f}_k(v-u) = \sum_{i=0}^{\lfloor v-u\rfloor-1} B_k[i] + (v-u-\lfloor v-u\rfloor)\cdot B_k[\lfloor v-u\rfloor].$$

The packing problem can be simply solved by merging the two non-decreasing arrays A_k and B_k into a single non-decreasing array and picking the first volume-v objects from the beginning, while the u value that achieves this minimum is precisely the volume of objects picked from array A_k.

Thus, let H_k be the non-decreasing array merged from A_k and B_k, then

$$g_k(v) = \sum_{i=0}^{\lfloor v\rfloor-1} H_k[i] + (v - \lfloor v\rfloor)\cdot H_k\big[\lfloor v\rfloor\big].$$

By defining $G_k[0] = H_k[0]$ and $G_k[i] = H_k[i] - H_k[i-1]$ for all positive integers i, we can re-write the above formula as $g_k(v) = \sum_{i=0}^{v} G_k[i]\cdot\langle v-i\rangle$, where $G_k[i]$ is always non-negative. In addition, by assumption, $F_k[0] = 0, F_k[1] = r+1, F_k[2] = 1$, so $A_k = [1, r+2, r+3, \dots]$ and $B_k = [r, 2r+1, 2r+2, \dots]$, hence we can compute $G_k[0] = H_k[0] = A_k[0] = 1$ ($r \ge 1 \Rightarrow A_k[0] \le B_k[0]$), $G_k[1] = H_k[1] - H_k[0] = B_k[0] - 1 = r-1$, $G_k[2] = H_k[2] - H_k[1] = A_k[1] - r = r + 2 - r = 2$. Therefore, Claim 4 holds. □

Last, we note that $u_n(v)$ can be computed as the volume of objects from A_n in the first v volume of H_n. A decimal v implies partially packing an object, and the partially packed volume is counted into $u_n(v)$ if and only if it comes from A_n. It is easy to see that $u_n(v)$ is a continuous piece-wise linear function, whose horizontal line-segments are those $[i, i+1]$ where $H_n[i]$ comes from B_n; and whose slope-1 line-segments are those $[i, i+1]$ where $H_n[i]$ comes from A_n.

D Proof of Claim 6

Claim 6. *The array G_n contains at most $O(n)$ non-zero entries.*

Proof. According to Claim 4, $G_n[i] \geq 0$ and $G_n[i] \in \mathbb{Z}[r]$ for all $i \in [0, 2^n - 1]$. Moreover, we know $\sum_{i=0}^{2^n-1} G_n[i] = H_n[2^n - 1]$ and by Lemma 14, $H_n[2^n - 1] = 1 + n \cdot r$. Let ℓ be the lower bound of all non-zero entries in G_n, then there can't be more than $\lceil (1 + n \cdot r)/\ell \rceil \in O(n)$ non-zero entries in G_n, otherwise their sum, $\sum_{i=0}^{2^n-1} G_n[i]$, will be greater than $1 + n \cdot r$. □

Lemma 14. *For all $n \geq 1$, $H_n[2^n - 1] = 1 + n \cdot r$.*

Proof. We will prove this lemma by induction. We have shown that $H_1[1] = 1 + r$ at the end of the proof of Lemma 13, hence the lemma holds when $n = 1$. Assume it holds when $n = k$. Recall that H_{k+1} is an array merged from array $\left\{1 + \sum_{i=0}^{j} F_{k+1}[i]\right\}_{0 \leq j < 2^k}$ and $\left\{r + \sum_{i=0}^{j} F_{k+1}[i]\right\}_{0 \leq j < 2^k}$. Since $r \geq 1$, we know

$$
\begin{aligned}
H_{k+1}[2^{k+1} - 1] &= r + \sum_{i=0}^{2^k-1} F_{k+1}[i] \\
&= r + F_{k+1}[0] + F_{k+1}[1] + F_{k+1}[2] + \sum_{i=3}^{2^k-1} F_{k+1}[i] \\
&= r + 0 + (r + 1) + 1 + \sum_{i=3}^{2^k-1} G_k[i] \\
&= r + 1 + (r - 1) + 2 + \sum_{i=3}^{2^k-1} G_k[i] \\
&= r + G_k[0] + G_k[1] + G_k[2] + \sum_{i=3}^{2^k-1} G_k[i] \\
&= r + \sum_{i=0}^{2^k-1} G_k[i] \\
&= r + H_k[2^k - 1] = r + 1 + k \cdot r = 1 + (k + 1) \cdot r.
\end{aligned}
$$

which shows the induction step, hence completes the proof. □

E Proof of Claim 8

Claim 8. *Let x^*_{a} be the probability of Algorithm 1 outputting string a, then $\{x^*_{\mathsf{a}}\}_{\mathsf{a} \in \{0,1\}^n}$ solves the linear program of Figure 2. Moreover, if $\varepsilon = 2^{-k}$ for some integer k, then $x^*_{\mathsf{a}} \in \{0, 2^{-k}\}$ for all $\mathsf{a} \in \{0, 1\}^n$.*

Proof. Let y^*_{a} be the value of y_{a} that solves the LP of Figure 4, hence $y^*_{\mathsf{a}} = \varepsilon x_{\mathsf{a}}$. Let f_n be the solution function to the LP of Figure 4 as was defined in Section 3.2. For every t $(1 \leq t \leq n)$, recall that $f_{n-t+1}(x)$ denotes the minimal extra cost of using the rest $n - t$ challenges bits if the probability of using t or more challenges is $x \cdot \varepsilon$. In the proof of Lemma 12, we have shown that

$$
f_{n-t+1}(x) = \begin{cases} x & 0 \leq x < 1 \\ x + (r + 1)(x - 1) & 1 \leq x < 2 \\ x + g_{n-t}(x) & x \geq 2 \end{cases} \tag{8}
$$

which can actually be achieved by setting (in the inductive step)

$$y_0 = \begin{cases} x & 0 \le x < 1 \\ 2 - x & 1 \le x < 2 \\ 0 & x \ge 2 \end{cases} \tag{9}$$

Note in the step of calculating f_{n-t+1}, εy_0 is the probability that a ends up having exactly t bits while $\mathsf{a}_t = \mathtt{0}$. y_0's formula above implies

1. If $0 \le x < 1$, then $\mathsf{len}(\mathsf{a}) = t$ (namely $\mathcal{C}$ uses precisely t challenge bits) happens with probability $x/x = 1$.

2. If $1 \le x < 2$, then: (1) with probability $(2-x)/x$, $\mathsf{len}(\mathsf{a}) = t$; (2) with probability $(x-1)/x$, $\mathsf{len}(\mathsf{a}) > t$ and $\mathsf{a}_{t+1} = \mathtt{0}$; (3) with probability $(x-1)/x$, $\mathsf{len}(\mathsf{a}) > t$ and $\mathsf{a}_{t+1} = \mathtt{1}$. Note that (2) and (3) can be derived from solving $g_{n-t}(2)$.

3. If $x > 2$, then $\mathsf{len}(\mathsf{a}) > t$. Moreover, the probability that $\mathsf{a}_{t+1} = \mathtt{0}$ will be $u_{n-t}(x)/x$, which has been shown in the proof of Lemma 13 when solving $g_{n-t}(x)$.

Now it is straightforward to verify that the code inside the **while** loop (line 3 of Algorithm 1) reflects precisely the theoretical analysis above.

Showing $x^*_{\mathsf{a}} \in \{0, 2^{-k}\}$ for all $\mathsf{a} \in \{\mathtt{0},\mathtt{1}\}^n$ is equivalent to proving $y^*_{\mathsf{a}} \in \{0,1\}$ for all $\mathsf{a} \in \{\mathtt{0},\mathtt{1}\}^n$ since $y^*_{\mathsf{a}} = \varepsilon x_{\mathsf{a}} = 2^k x_{\mathsf{a}}$. Following formula (9), if x is an integer, the y_0 must be either 0 or 1. Then it reduces to showing that the x values on all layers of recursion are an integer, i.e., all recursive calls to $f_n, f_{n-1}, \ldots, f_1$ are with integer inputs. By formula (8), if f is invoked with an integer x, then the next g will also be invoked with integer x. By Claim 2, we know that if g is invoked with an integer v, then f will be invoked with u and $v-u$, both of which are integer because f_n is a piece-wise linear function. Finally, note that f_n is initially invoked with integer $x = 2^k$, thus, all subsequent invocation of g and f are with integer values. This completes the proof that $y^*_{\mathsf{a}} \in \{0,1\}$. □

F Proofs of Global Optimality

Claim 9. *If there exist $\hat{n}$ such that $G_{\hat{n}}[i] = G_{\hat{n}+1}[i]$ for all $0 \le i < 2^{\hat{n}}$, then $g_{\hat{n}}(v) = g_{\hat{n}+i}(v)$ for all positive integer i and real v $(0 < v \le 2^{\hat{n}})$.*

Proof. Fix an $\varepsilon \ge 1/2^n$. We know that $g_n(1/\varepsilon)$ can be deterministically computed from $\{G_n[i] \,|\, 0 \le i < 2^n\}$ for all n (by Claim 4), which, in turn, is deterministically computable from $\left\{F_{n-1}[i] \,|\, 0 \le i < 2^{n-1}\right\}$ (by Lemma 13), which, again, can be deterministically computed from $\left\{G_{n-1}[i] \,|\, 0 \le i < 2^{n-1}\right\}$ (by Lemma 12). Therefore, once the first 2^n entries of $G_n[i]$ is identical to those of G_{n+1}, G_n stops changing as n increases, hence g_n also stops changing. □

Claim 10. *For all ε $(0 < \varepsilon < 1)$, let $\hat{n}$ $(\hat{n} \ge \lceil \varepsilon \rceil)$ be the smallest integer such that $G_{\hat{n}}[i] = G_{\hat{n}+1}[i]$ for all $0 \le i < 2^{\hat{n}}$, and $g_\infty(1/\varepsilon)$ be the minimal cost of the optimal strategy that allows infinite-length challenges, then $g_{\hat{n}}(1/\varepsilon) \le g_\infty(1/\varepsilon)$.*

Proof. Fix ε. Aiming at a contradiction, we assume $g_\infty(1/\varepsilon) < g_{\hat{n}}(1/\varepsilon)$ and let S be an optimal strategy that achieves $g_\infty(1/\varepsilon)$. Let S_1 be an optimal strategy that achieves $g_{\hat{n}}(1/\varepsilon)$. Since $g_n(1/\varepsilon)$ is a continuous function of ε for all n, for an $\varepsilon_2 > \varepsilon$, there is an optimal mixed-strategy S_2 (using only finite-length challenges) whose expected cost lies in between $g_\infty(1/\varepsilon)$ and $g_{\hat{n}}(1/\varepsilon)$. On the other hand, we can derive a strategy S_3 from S by simply giving up all strategies that use challenges

longer than some ℓ. Thus, S_3 has an expected cost less than $g_\infty(1/\varepsilon)$ but offers a weaker security guarantee (i.e., its failure probability $\varepsilon_3 > \varepsilon$). Note that ℓ decreases monotonically as ε_3 increases and for every ε_3, we can always find an ℓ for every ε_3. Now if we pick $\varepsilon_3 = \varepsilon_2$, a particular strategy S_3 obtained this way, using only finite number of cut-and-choose bits, will have an expected cost less than $g_\infty(1/\varepsilon)$, which is in turn less than the expected cost of S_2, hence a contradiction with the fact that S_2 is optimal among all strategies using finite-length challenges. □

G Efficient Secure Computation using Variable-Length Challenges

Let Hash be a collision resistant hash function, REHash be a collision-resistant hash function that is also a suitable *randomness extractor* [10], and Com be a commitment scheme. We will use Hash to commit garbled circuits succinctly. We use REHash to create the generator's input wire labels from ElGamal commitments. We use Com to commit the preimage of the generator's input wire labels.

G.1 Building Blocks

Garbled Circuits Yao's garbled-circuit provides a generic mechanism to construct a (passively) secure two-party protocol for computing f starting from any boolean circuit for f [22]. Bellare, Hoang, and Rogaway [3] formalized a *garbling scheme* $\mathcal{G}$ as a 5-tuple $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}, f)$ of algorithms, where Gb is an efficient randomized *garbler* that, on input $(1^k, f)$, outputs (F, e, d); En is an *encoder* that, on input (e, x), outputs X; Ev is an *evaluator* that, on input (F, X), outputs Y; De is a *decoder* that, on input (d, Y), outputs y. The correctness of $\mathcal{G}$ requires for every $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$ and every x, $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = f(x)$. Let "$\approx$" denote computational indistinguishability. *Privacy* of $\mathcal{G}$ implies that there exists an efficient simulator $\mathcal{S}$ such that for any $x \in \{0,1\}^\ell$,

$$\left\{(F, e, d) \leftarrow \mathsf{Gb}(1^k, f), X \leftarrow \mathsf{En}(e, x) : (F, X, d)\right\} \approx \{\mathcal{S}(1^k, f, f(x, y))\}.$$

In a garbled-circuit protocol, one party (the circuit *generator*) prepares an "encrypted" version of a circuit computing f. The second party (the circuit *evaluator*) then obliviously computes the output of the circuit without learning any intermediate values. Starting with a (known) boolean circuit for f, the circuit generator associates two random cryptographic keys w_i^0, w_i^1 with each wire i of the circuit, where w_i^0 encodes a 0-bit and w_i^1 encodes a 1-bit. Then, for each binary gate g of the circuit with input wires i, j and output wire k, the generator computes ciphertexts

$$\mathrm{Enc}_{w_i^{b_i}}\left(\mathrm{Enc}_{w_j^{b_j}}\left(w_k^{g(b_i,b_j)}\right)\right)$$

for $b_i, b_j \in \{0,1\}$. The resulting four ciphertexts, in random order, constitute a *garbled gate*. The collection of all garbled gates forms the *garbled circuit* that is sent to the evaluator. Given keys w_i, w_j associated with both input wires i, j of some garbled gate, the evaluator can compute a key for the output wire of that gate by decrypting the appropriate ciphertext. These keys can then be mapped to their semantic values using mappings provided by the circuit generator.

Oblivious Transfer An *oblivious transfer* (OT) protocol allows a *sender*, holding strings w^0 and w^1, to transfer w^b to a receiver holding a selection bit b; the receiver learns nothing about w^{1-b}, and the sender does not learn b. A *committing oblivious transfer* (COT) gives the receiver additionally the commitments of the sender's inputs. We use the batched 1-out-of-2 COT as a black-box to send wire labels corresponding to the evaluator's input. UC secure 1-out-of-2 committing OT can be efficiently instantiated from dual-mode cryptosystems [19].

ElGamal Commitment Let g be a generator of a group $\mathbb{G}$ of order q where DDH is assumed to be hard and $h \in \mathbb{G}$ be an element whose discrete logarithm (base-g) is hidden to the commitment receiver. To commit a message m using ElGamal commitment, the committer sends $\mathsf{EGCom}(h; m, r) = (g^r, h^r g^m)$. To open an commitment, the committer sends r. ElGamal commitment is perfectly binding, as well as computationally hiding. More importantly, we use the following two properties of ElGamal commitment in our protocol:

1. Given $\mathsf{EGCom}(h; m, r)$ and $\mathsf{EGCom}(h; m', r')$, it is possible to prove $m = m'$ in zero-knowledge using the ZKPoK for DH-tuples (also used by Lindell [18]).

2. Given $\mathsf{EGCom}(h; m, r)$, $m \in \{0, 1\}$, and $w = \log_g h$, it is easy to learn m, since $m = 0$ if and only if $[\mathsf{EGCom}(h; m, r)]_1 = [\mathsf{EGCom}(h; m, r)]_0^w$, where $[\cdot]_i$ refers to the i^{th} component of a tuple.

G.2 Protocol Specification

We assume the inputs of P_1 and P_2 are x and y ($x, y \in \{0,1\}^\ell$), respectively, and the output $f(x, y) \in \{0, 1\}$. We assume P_1 is the circuit generator and P_2 the evaluator. Let $\mathbb{G} = \langle g \rangle$ be a multiplicative group of order q where DDH is assumed to be hard. We use "$s \xleftarrow{seed_j} S$" to denote a uniform sampling from the set S using randomness drawn from $seed_j$, (as opposed to "$\leftarrow$" that refers to sampling using an unspecified randomness). The protocol proceeds as follows.

1. P_1 and P_2 run a committing coin-tossing protocol where P_2 learns the output and P_1 learns the commitment of the output. P_2 runs Algorithm 1 with the coin-tossing output to pick its challenge-string a.

2. P_1 picks $w \leftarrow \mathbb{Z}_q$, $w_0 \leftarrow \mathbb{Z}_q$, and sets $w_1 = w - w_0 \mod q$. It then sends $h = g^w$, $h_0 = g^{w_0}$ and $h_1 = g^{w_1}$ to P_2, who verifies $h = h_0 \cdot h_1$.

3. P_1 keeps generating garbled circuits of f until being notified to stop. For the j-th garbled circuit:

   (a) P_1 generates the whole circuit from a randomly picked secret $seed_j$. In particular,

      i. For the i-th input wire for P_1's input, P_1 picks $r_{i,j}^0 \xleftarrow{seed_j} \mathbb{Z}_q$ and $r_{i,j}^1 \xleftarrow{seed_j} \mathbb{Z}_q$, and computes $u_{i,j}^0 \coloneqq \mathsf{EGCom}(h; 0, r_{i,j}^0)$ and $u_{i,j}^1 \coloneqq \mathsf{EGCom}(h; 1, r_{i,j}^1)$, where $\mathsf{EGCom}(h; m, r) = (g^r, h^r \cdot g^m)$. Then P_1 sets the two wire labels $X_{i,j}^0 \coloneqq \mathsf{REHash}(u_{i,j}^0)$ and $X_{i,j}^1 \coloneqq \mathsf{REHash}(u_{i,j}^1)$.

   (b) For all $1 \le i \le \ell$, P_1 computes and randomly permutes $\mathsf{Com}(u_{i,j}^0)$ and $\mathsf{Com}(u_{i,j}^1)$, then sends them to P_2.

   (c) P_1 picks two wire labels for the final output wire: $Z_j^0 \xleftarrow{seed_j} \mathbb{Z}_q$, $Z_j^1 \xleftarrow{seed_j} \mathbb{Z}_q$.

   (d) P_1 sends output recovery commitments $h_0 g^{Z_j^0}$ and $h_1 g^{Z_j^1}$.

   (e) P_1 computes the hash of the garbled circuit, $hash_j$, and sends it to P_2.

4. P_2 receives $\mathsf{len}(\mathsf{a})$ garbled circuit hashes (along with their output recovery commitment pairs) and notifies P_1 to stop. Then, P_2 opens the coin-tossing result to P_1, who then derives P_2's the challenge-string a.

5. P_1 and P_2 run an OT protocol for every bit of y, where P_1, as the OT sender, inputs $\left\{Y_{i,j}^0\right\}_{j=1}^{\mathsf{len(a)}}$ and $\left\{Y_{i,j}^1\right\}_{j=1}^{\mathsf{len(a)}}$; P_2, as the OT receiver, inputs y_i (the i^{th} bit of y). At the end of this step, P_2 learns $\left\{Y_{i,j}^{y_i} \mid 1 \leq i \leq \ell, 1 \leq j \leq \mathsf{len(a)}\right\}$.

6. For all $1 \leq j \leq \mathsf{len(a)}$, P_2 selects the j-th circuit to be checked if $\mathtt{a}_j = \mathtt{0}$, hence requesting P_1 to reveal $seed_j$. P_2 checks the following:

   (a) The circuit generated from $seed_j$ matches with its hash $hash_j$ received in step 3e.

   (b) The wire labels for each of P_1's input wire match with their commitments received in step 3b under some permutation.

   (c) The OT inputs of P_1 match with the input labels used in every check-circuit (thus we require Committing OT).

   If any circuit is found incorrect, P_2 aborts.

7. For all $1 \leq j \leq \mathsf{len(a)}$, P_2 selects the j-th circuit if $\mathtt{a}_j = \mathtt{1}$, and evaluate it as follows:

   (a) For every $1 \leq i \leq \ell$, P_1 decommits one of the two commitments sent in step 3b for P_2 to learn $u_{i,j}^{x_i}$. P_2 computes $X_{i,j}^{x_i} = \mathsf{REHash}(u_{i,j}^{x_i})$.

   (b) P_1 sends $w_0 + Z_j^0$ and $w_1 + Z_j^1$ to P_2, who verifies that they are consistent with the output recovery commitments received in step 3d.

   (c) P_2 evaluates the circuit using $\left\{X_{i,j}^{x_i}\right\}_{i=1}^{\ell}$ and $\left\{Y_{i,j}^{y_i}\right\}_{i=1}^{\ell}$, obtaining $\hat{Z}_j$.

8. For all $i \in [1, \ell]$, P_1 proves (using efficient DH-based ZKPoK [18, Protocol 3.2, Step 9]) that $\left\{u_{i,j}^{x_i} \mid \mathtt{a}_j = \mathtt{1}\right\}$ are consistently associated with the same x_i bit. That is, P_1 proves in zero-knowledge that

$$\text{Either } \left(\forall j \in \{j \mid \mathtt{a}_j = \mathtt{1}\} : \left(g, g^{r_{i,j}^0}, h, \left[u_{i,j}^{x_i}\right]_1\right) \in DH\right)$$
$$\text{or } \left(\forall j \in \{j \mid \mathtt{a}_j = \mathtt{1}\} : \left(g, g^{r_{i,j}^1}, h, \left[u_{i,j}^{x_i}\right]_1 / g\right) \in DH\right).$$

   Recall that $u_{i,j}^{x_i} = \left(g^{r_{i,j}^{x_i}}, h^{r_{i,j}^{x_i}} g^{x_i}\right)$ and $\left[\,\cdot\,\right]_i$ refers to the i^{th} entry in a tuple.

9. Note that exactly one of the following three cases must happen.

   (a) If there exist $j', j'' \in \{j \mid \mathtt{a}_j = \mathtt{1}\}$ such that $\hat{Z}_{j'}$ matches up with $h_0 g^{Z_{j'}^0}$ and $\hat{Z}_{j''}$ matches up with $h_1 g^{Z_{j''}^1}$, then P_2 uses $\hat{Z}_{j'}$ and $\hat{Z}_{j''}$ to recover x (through recovering both w_0 and w_1, hence w, which allows to open $u_{i,j}^{x_i} = \mathsf{EGCom}(h; x_i, r)$ for all $i \in [1, \ell]$), then outputs $f(x, y)$.

   (b) If for all $j \in \{j \mid \mathtt{a}_j = \mathtt{1}\}$, there exists a unique b for all matching $\hat{Z}_j$ and $h_b g^{Z_j^b}$, then P_2 outputs b.

   (c) If there does not exist any j such that $\hat{Z}_j$ matches with $h_b g^{Z_j^b}$, then P_2 aborts.

G.3 Proof of Security

Theorem 15. *Under the assumptions outlined in Section G.1, the protocol given in Section G.2 securely computes f in the presence of malicious adversaries.*

We consider the standard definition of security for actively secure two-party computation allowing aborts [12, Section 7.2.3]. In our proof, we refer to the term "negligible" in the concrete (instead of asymptotic) sense, i.e., any non-negative value smaller than ε is considered negligible.

Proof. We prove our protocol in Section G.2 secure in a hybrid world where the parties have access to ideal functionalities for the committing coin-tossing and oblivious transfer. By standard composition theorems [8], this implies security when those subroutines are instantiated using secure protocols for those tasks.

Corrupted P_1: For every corrupted P_1, we describe an efficient simulator $\mathcal{S}_1$ (running the corrupted P_1 as a subroutine) that interacts with an honest P_2 through a trusted party T computing f. We shall show that the joint distribution of the outputs of $\mathcal{S}_1$ and P_2 is computationally indistinguishable from that of P_1 interacting with P_2 in the real world execution. $\mathcal{S}_1$ works as follows:

1. $\mathcal{S}_1$, pretending an honest P_2 with input $y = 0$, runs P_1 till step 8 (if P_1 fails any of the checks during the process, $\mathcal{S}_1$ sends $\perp$ to the trusted party and outputs whatever P_1 outputs).

2. If P_1 responds correctly, $\mathcal{S}_1$ extracts P_1's effective input x through the zero-knowledge proof of knowledge proof. $\mathcal{S}_1$ sends x to the trusted party, and receives $f(x, y)$ in return. $\mathcal{S}_1$ gives $f(x, y)$ to P_1 and outputs whatever P_1 outputs.

The fact that the above experiment is indistinguishable from that of running P_1 and P_2 in the real world can be proved through examining the behavior of $\mathcal{S}_1$ in every step:

1. In step 1 of $\mathcal{S}_1$, if $\mathcal{S}_1$ ends up sending $\perp$ to T and outputting P_1's outputs, exactly the same thing will happen when P_2 executes in the real world because $\mathcal{S}_1$ runs the same instructions as a real world P_2 does (except that $\mathcal{S}_1$ always sets $y = 0$, a specific difference that is guaranteed to be hidden to P_1 by the security of oblivious transfer). If P_1 responds correctly to all the checks in step 1, $\mathcal{S}_1$ proceeds to step 2 and we analyze further below.

2. In step 2 of $\mathcal{S}_1$, since P_1 passes all the checks, the ideal world P_2 will output $f(x, y)$, while the real world P_2 will also be able to recover x (with the same input recovery mechanism proposed by [1]), except for a negligible probability.

We additionally point out that a malicious P_1 can't hope distinguish the two experiments by behaving differently depending on the cut-and-choose string of $\mathcal{S}_1$. This is because

1. P_1 can win $\mathcal{S}_1$ in cut-and-choose with only negligible probability, as $\mathcal{S}_1$ sampled its cut-and-choose string with Algorithm 1.

2. Assuming P_1 does not win $\mathcal{S}_1$ in the cut-and-choose game, a malicious P_1 who responds incorrectly to $\mathcal{S}_1$ in all polynomial runs (with different random tapes of $\mathcal{S}_1$) is destined to respond incorrectly to P_2 in the real world, except for a negligible probability.

Corrupted P_2: For every corrupted P_2, we describe an efficient simulator $\mathcal{S}_2$ (running the corrupted P_2 as a subroutine) that interacts with an honest P_1 through a trusted party (T) computing f. We shall show that the joint distribution of the outputs of P_1 and $\mathcal{S}_2$ is computationally indistinguishable from that of P_1 interacting with P_2 in the real world execution. $\mathcal{S}_2$ works as follows:

1. $\mathcal{S}_2$ runs P_2 till step 1 (coin-tossing), where $\mathcal{S}_2$ extracts P_2's cut-and-choose string a through the ideal coin-tossing functionality.

2. $\mathcal{S}_2$, pretending an honest P_1 with $x = 0$, runs P_2 till step 5. At step 5, $\mathcal{S}_1$ extracts P_2's effective input y through the ideal functionality for OT. $\mathcal{S}_2$ sends y to T and in return receives $f(x, y)$. Then, $\mathcal{S}_2$ rewinds P_2.

3. $\mathcal{S}_2$, pretending P_1 with input $x = 0$, runs P_2 till step 3. At step 3, for every check circuit, P_2 generates a garbled circuit honestly; while for every evaluation circuit, P_2 invokes the garbled circuit simulator (the one used in defining garbling scheme *privacy*) with f and $f(x, y)$ and sends the simulated garbled circuit to P_2. (Note that $\mathcal{S}_2$ already learns P_2's cut-and-choose string a from the rewound coin-tossing step.)

   $\mathcal{S}_2$ resumes P_2 till the end and outputs whatever P_2 outputs.

The indistinguishability of this experiment and the one running P_1 and P_2 in the real world can be easily derived from the security (more specifically, the notion of *privacy*) of the garbling scheme. □

H Cost-Aware Prefix-free Codes

Algorithm 2 $\mathsf{encode}(w; n, r)$

Require: $\varepsilon \in \mathbb{R}^+$, $n \in \mathbb{Z}^+$ and $r > 1$, $w \in \{0,1\}^n$.

Pre-compute $\{u_k \mid 1 \le k \le n\}$ from r as shown at the end of the proof of Lemma 13
$x \coloneqq 2^{\mathsf{len}(w)}$; $k \coloneqq n$; $i \coloneqq 1$
while $x \ge 1$ **do**
  if $x = 1$ **then**
    return a
  else
    if $w < u_k(x)$ **then**
      $\mathsf{a}_i \coloneqq 0$
      $x \coloneqq u_k(x)$
    else
      $\mathsf{a}_i \coloneqq 1$
      $w \coloneqq w - u_k(x)$
      $x \coloneqq x - u_k(x)$
  $k \coloneqq k - 1$; $i \coloneqq i + 1$

Algorithm 3 $\mathsf{decode}(\mathsf{a}; n, r)$

Require: $\varepsilon \in \mathbb{R}^+$, $n \in \mathbb{Z}^+$, $r > 1$ and a.

Pre-compute $\{u_k \mid 1 \leq k \leq n\}$ from r as shown at the end of the proof of Lemma 13
$x \coloneqq 1; \quad \ell = \mathsf{len}(\mathsf{a}); \quad k \coloneqq n - \ell + 1; \quad i \coloneqq \ell; \quad w \coloneqq 0$
while $i \geq 0$ **do**
    if $i = 0$ **then**
        return w
    else
        if $\mathsf{a}_i = 0$ **then**
            $x \coloneqq u_k(x)$
        else
            $w \coloneqq w + u_k(x)$
            $x \coloneqq x + u_k(x)$
    $k \coloneqq k + 1; \quad i \coloneqq i - 1$