Searching for Cryptogenography Upper Bounds via Sum of Square Programming

Dominik Scheder
Shanghai Jiao Tong University, China
dominik@cs.sjtu.edu.cn

Shuyang Tang
Shanghai Jiao Tong University, China
htftsy@sjtu.edu.cn

Jiaheng Zhang
UC Berkeley, USA
jiaheng_zhang@berkeley.edu

Abstract

Cryptogenography is a secret-leaking game in which one of n players is holding a secret to be leaked. The n players engage in communication as to (1) reveal the secret while (2) keeping the identity of the secret holder as obscure as possible. All communication is public, and no computational hardness assumptions are made, i.e., the setting is purely information theoretic. Brody, Jakobsen, Scheder, and Winkler [2] formally defined this problem, showed that it has an equivalent geometric characterization, and gave upper and lower bounds for the case in which the n players want to leak a single bit. Surprisingly, even the easiest case, where two players want to leak a secret consisting of a single bit, is not completely understood. Doerr and Künnemann [4] showed how to automatically search for good protocols using a computer, thus finding an improved protocol for the 1-bit two-player case. In this work, we show how the search for upper bounds (impossibility results) can be formulated as a Sum of Squares program. We implement this idea for the 1-bit two-player case and significantly improve the previous upper bound from $47/128 = 0.3671875$ to 0.35183.

2012 ACM Subject Classification Theory of computation → Communication complexity; Theory of computation → Semidefinite programming

Keywords and phrases Communication Complexity, Secret Leaking, Sum of Squares Programming

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.31

Supplement Material `http://basics.sjtu.edu.cn/~dominik/sos-cryptogenography/`

Funding *Dominik Scheder*: This research has been supported by the National Natural Science Foundation of China under grant 61502300.

1 Introduction

Cryptogenography is a whistleblowing problem involving n players. One player $J \in [n]$ is holding a secret $X \in \mathcal{X}$. The value of X is unknown to everybody else. The value of J is unknown, too (the other players just know that they do not have the secret). The goal of the players is to publish the secret while keeping the identity of J as obscure as possible. To achieve this goal, the players engage in public communication according to some publicly known protocol. The protocol ends once the value of the secret has been determined. At this point, the authorities step in and arrest the prime suspect, i.e., the player with the highest a-posteriori probability to be J.

In an (imaginary) application, the secret X could be a compromising document of some government agency or company, J a whistleblower, and $[n]$ a community of transparency activists of which J is a member. A more mundane application sees X as a pirated movie, J

as the person sharing the video, and $[n]$ as the users of a file sharing platform. In both cases, the participants want to keep the leaker J anonymous but make the secret X public. This distinguishes it from most other problems in cryptography.

Towards a rigorous formulation, suppose (J, X) are drawn from some *prior distribution* D on $[n] \times \mathcal{X}$. Typically, this will be the uniform distribution, representing a complete lack of knowledge about X and J. A communication protocol is a finite binary tree T, in which every inner node u is labeled with a speaker $P_u \in [n]$ and a vector $(p_i)_{i \in \mathcal{X} \cup \{*\}}$. The semantics of this tree is as follows: we start at the root of the tree. At an inner node u, player P_u sends the next bit of communication. If player P_u is the secret holder, she sends `right` with probability p_X (and `left` with probability $1 - p_X$). If she does not hold the secret, her message cannot depend on X, and thus she sends `right` with probability p_*. The protocol then proceeds to the left or right child of u, depending on the message sent by P_u.

With every node u of the tree we associate a distribution D_u on $[n] \cdot \mathcal{X}$. Namely, $D_u(i, x)$ is the probability that player i holds the secret and the secret is x, conditioned on the protocol arriving at node u. Note that D_{root} is the prior distribution. A protocol is said to be *valid* if X is determined once the protocol ends. In other words, for every leaf l of the protocol tree there is some $x \in \mathcal{X}$ such that $\sum_i D_l(i, x) = 1$. We call x the *output* of leaf l. If the protocol reaches leaf l, an outside observer can be sure that the secret is x. At this point, the authorities arrest the prime suspect, which is the player i maximizing $D_l(i, x)$. The players win if the arrested player is not the secret holder, which happens with probability $1 - \max_{i \in [n]} D_l(i, x)$. The *value* $\text{val}(u)$ of a node u, is the winning probability of the players, conditioned on this node being reached (so $\text{val}(l) = 1 - \max_i D_l(i, x)$ for a leaf l with output x, and $\text{val}(u)$ is a weighted average of $\text{val}(u_0)$ and $\text{val}(u_1)$, where u_0 and u_1 are the two children of u). The value of the protocol is the value of its root (which is equal to the overall winning probability). The value $\text{val}(D)$ of a distribution D is the supremum of $\text{val}(\text{root})$, taken over all valid protocol trees. In words, it is the optimal achievable success probability for the Cryptogenography problem with prior distribution D.

Note that our scenario is purely information theoretical. We assume participants and the authorities to be computationally unlimited. This precludes us from using established methods like public-key cryptography. Also, all communicated is public, and thus methods from multi-party communication (which otherwise could easily solve this problem) do not apply here. In general, we consider cryptogenography as being part of *communication complexity*, and we study it to explore the limitations of randomized two-party (or multi-party) communication.

1.1 Previous Work

The Cryptogenography problem was introduced by Brody, Jakobsen, Scheder, and Winkler [2]. They also chose the name Cryptogenography, which roughly translates to "hidden source writing". This is because we want to protect the source and not, as would be more common in cryptography, the message. They show that finding the optimal value is equivalent to optimizing a function on the simplex $\Delta_{[n] \times \mathcal{X}}$ (the set of probability distributions on $[n] \times \mathcal{X}$), subject to certain concavity constraints (see Section 2). As for concrete bounds, [2] focus on the case $|\mathcal{X}| = 2$, i.e., the secret consists of a single bit. This might feel a bit unrealistic, as most government documents and most movies consist of more than one bit; however, already the 1-bit case turns out to be challenging. For the n-player case, [2] show that a simple majority voting protocol has a success probability of 0.5406 if $n \geq 23$; a more sophisticated protocol, based on voting with abstention, achieves 0.5644 if $n \geq 1200$. They also prove an upper bound of $\frac{3}{4} - \frac{1}{2n}$. For $n = 2$, the simplest non-trivial case, they show a protocol achieving $1/3$ and an upper bound of $3/8$.

The success probability of $1/3$ for $n = 2$ is achieved by a simple three-round protocol. Somewhat surprisingly, it is not optimal. Doerr and Künnemann [4] showed how Cryptogenography protocol design can be viewed as a certain vector splitting game, and used a computer program to find improved protocols. Their best protocol tree consists of 18248 nodes and achieves a success probability of 0.3384; the shortest protocol (in terms of rounds) they found which beats $1/3$ uses up to sixteen rounds of communication. They also improve the upper bound for $n = 2$ from $3/8$ to $\frac{47}{128}$.

What about the general case of larger $n, |\mathcal{X}| > 2$? Sune Jakobsen [5] showed what the skeptical reader might already suspect: as $|\mathcal{X}|$ increases, the optimal success probability goes to 0, regardless of n. Thus, secure secret leakage is impossible, purely information theoretically speaking. In fact, Jakobsen considered an even more general setting, in which $k = \gamma n$ "insider players" know the secret, and $\mathcal{X} = \{0,1\}^b$, i.e., the secret has b bits, with $b = \beta n$. He gives a simple and beautiful protocol using error correcting codes and Shannon's Noisy Channel Coding Theorem (see for example [3]) to show that the players have a reasonable success probability if β is not too large compared to γ. He also shows that this protocol is in some sense asymptotically optimal. Unfortunately, his method give exact results only if γ, β are constants and n tends to infinity and thus do not help us for the case $|\mathcal{X}| = 2$.

Furthermore, Jakobsen and Orlandi [6] introduce a model in which a *almost all* communication is public, and only a very small key is sent through an anonymous channel. They show that with high probability, the secret leaker stays anonymous. However, they assume the adversary to be computationally bounded, and thus their techniques do not apply to our purely information theoretic setting.

1.2 Our Contribution

Like Doerr and Künnemann [4], we focus on the 1-bit 2-player case: $n = 2$ and $\mathcal{X} = \{0,1\}$. We show how to search for *upper bounds* in an automated way. We use the geometric characterization of [2], which states that to find an upper bound on the possible success probability, it is enough to minimize a function $f : \Delta_{[n]\times\mathcal{X}} \mapsto \mathbb{R}$ subject to certain boundary and concavity constraints. Since $n = 2$ and $\mathcal{X} = \{0,1\}$, the function f has only four variables. If we let f be a degree-d polynomial in those four variable, this becomes an optimization problem *in the coefficients of* f. We make this optimization problem tractable by formulating it as a Sum-of-Squares problem (SoS), which we then solve using the Matlab packages `yalmip`[1] [9] and `sostools` [10]. The following table summarizes our results obtained with `yalmip`. The numerical error arising from the SoS solver needs careful treatment in our case, and indeed causes degree 8 to be *worse* than degree 6.

Degree	SoS opt	numerical error	SoS opt + error	Remark
2	0.375	0	0.375	Same as [2]
4	0.3644	negligible	0.3644	slightly better than 0.3671875 from [4]
6	0.351612	0.000217	0.35183	our best bound so far
8	0.349515	0.0079587	0.35747	worse than degree 6 (numerical errors)

In theory, our techniques easily extend to the multi-player case and beyond the 1-bit case. However, the complexity increases in a way that makes it quickly impractical for current SoS solvers.

[1] `https://yalmip.github.io/`

2 Geometric Formulation

We now formally introduce the geometric characterization given by Brody, Jakobsen, Scheder, and Winkler [2]. We focus on the two-player 1-bit case, since this already contains all main ideas. Consider the simplex $\Delta := \{\text{Alice}, \text{Bob}\} \times \{0, 1\}$, the space of all probability distributions on who has the secret bit and what its value is. We represent a distribution $\mathcal{D}$ as a matrix:

	0	1
Alice	x_0	x_1
Bob	y_0	y_1

,

or short as $\begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix}$. Given a protocol tree, we associate a distribution $D_u \in \Delta$ with every node u: namely, $D_u(j, x)$ is the probability that j is the secret holder and x is the secret, conditioned on the protocol passing through u. Suppose it's Alice's turn to send a bit at node u. Let p be the probability that she sends `right` at node u and denote by D, D', and D'' the distributions at u, its left child, and its right child, respectively. Then $D = p\,D' + (1-p)\,D''$, so the three distributions

$$\begin{bmatrix} x'_0 & x'_1 \\ y'_0 & y'_1 \end{bmatrix}, \begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix}, \begin{bmatrix} x''_0 & x''_1 \\ y''_0 & y''_1 \end{bmatrix}$$

lie on a common line. In fact, this line has a crucial additional property: the line through $[y'_0, y'_1], [y_0, y_1], [y''_0, y''_1]$ also passes through the origin! In other words, the ratio between y_0 and y_1 is the same for all three distributions. This is because Alice can give hints about (1) whether she has the secret and (2) what its value is *provided she owns it* but *not* what the value is *if Bob has it*. This property is formally proven in [2]. The upshot is that the line through D, D', D'' can be parametrized as

$$\ell : t \mapsto \begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix} + t \begin{bmatrix} r_0 & r_1 \\ s_0 & s_1 \end{bmatrix} \tag{1}$$

where

$$x_0 + x_1 + y_0 + y_1 = 1 \tag{2}$$
$$x_0, x_1, y_0, y_1 \geq 0 \tag{3}$$
$$r_0 + r_1 + s_0 + s_1 = 0 \tag{4}$$
$$s_0\,y_1 - s_1\,y_0 = 0 \tag{5}$$
$$s_0\,s_1 \geq 0. \tag{6}$$

Note that (2) and (3) simply state that D is a distribution; (4) must hold since the line contains two other distributions D' and D'', too. Equality (5) states that the line $[y_0\ y_1] + t\,[s_0\ s_1]$ contains the origin. Finally, (6) follows from the other constraints and is included only because it turns out to help the SoS solver.

A line of the form (1) satisfying (2)–(6) is an *Alice-line*. If it's Bob's turn to talk at u, then equality (5) and (6) and should be replaced by $r_0\,x_1 - r_1\,x_0 = 0$ and $r_0\,r_1 \geq 0$; we call such a line a *Bob*-line. A *player-line* is a line that is an Alice-line or a Bob-line. A player-line is *non-trivial* if it intersects Δ in more than one point.

To wrap up, if Alice talks at node u, she "splits" the distribution D into D' and D'' such that all three distributions lie on an Alice-line. A certain converse of this actually holds, too: if D is the distribution associated with node u, and D lies between the distributions D' and D'' on a common Alice-line, then there is a way for Alice to sample her message (depending on whether she has the secret and if yes, what it is), such that the two children of u will be labeled with D' and D'', respectively. The (straightforward) proof can be found in [2].

▶ **Definition 1.** *A function $f : \Delta \to \mathbb{R}$ is called admissible if*

(a) $f(D) \geq 0$ *for* $D = \begin{pmatrix}1 & 0\\0 & 0\end{pmatrix}, \begin{pmatrix}0 & 1\\0 & 0\end{pmatrix}, \begin{pmatrix}0 & 0\\1 & 0\end{pmatrix}$, *and* $\begin{pmatrix}0 & 0\\0 & 1\end{pmatrix}$;

(b) $f(D) \geq 1/2$ *for* $D = \begin{pmatrix}1/2 & 0\\1/2 & 0\end{pmatrix}$ *and* $\begin{pmatrix}0 & 1/2\\0 & 1/2\end{pmatrix}$;

(c) *If ℓ is a non-trivial player-line, then $f|_\ell$, the restriction of f to ℓ, is concave at every point in Δ.*

The following theorem gives a geometric characterization of the problem. It is an adaptation of Lemma 4.3 from [2], together with the additional observation, made in [4], that the six "boundary distributions" in Point (a) and Point (b) actually suffice.

▶ **Theorem 2** (adapted from [2] and [4])**.** *If f is an admissible function, then $f(D) \geq \mathrm{val}(D)$ for all distributions D.*

The proof of the theorem is simple and works by induction on the protocol tree. Point (a) and Point (b) of Definition 1 serve as the induction base, and concavity along player-lines (c) is used in the induction step.

A similar geometric characterization by has been used by Mark Braverman, Ankit Garg, Denis Pankratov, and Omri Weinstein [1] to determine the information complexity of zero-error protocols computing the 2-bit AND function, and subsequently determining the asymptotic communication complexity of DISJOINTNESS, one of the most important functions in communication complexity.

▶ **Lemma 3.** *A function $f : \Delta \to \mathbb{R}$ is concave along every Alice-line (respectively, Bob-line), if and only if $\frac{\partial^2}{\partial t^2} f(\ell(t))|_{t=0} \leq 0$ for every Alice-line ℓ as in (1) (respectively, Bob-line).*

Proof. For fixed $x_0, x_1, y_0, y_1, r_0, r_1, s_0, s_1$ the function $g(t) = f(\ell(t))$ is a univariate polynomial in t. For the "if" part, not that if this is not concave for some t^* such that $\ell(t^*) \in \Delta$, then $g''(t^*) > 0$; furthermore, we can re-parametrize the line as $\ell(t^* + t)$; this is also an Alice line (resp., Bob-line), and concavity is violated at $t = 0$. For the "only if" part, note that concavity of $f|_\ell$ implies that $g''(0) \leq 0$. ◀

3 Sum of Square Programs

In this section we introduce the absolute basics of Sum of Square (SoS) Programs. SoS programming is a rapidly evolving field with vast literature but unfortunately no standard textbook yet. The reader interested in actually solving SoS programs may have a look at the SOSTOOLS User's guide [11]. If the reader is more interested in the theoretical background of SoS programming, they might start with Laurent [8].

How do you show that $P(x, y, z) = (x^2+y)^2\,(1+z^2) - 4\,xyz$ is non-negative? For example, you could check that $P(x, y, z) = (x - yz)^2 + (y - xz)^2$. In other words, you write it as a *sum of squares* polynomial (SoS). Next, consider $Q(x, y) = 1 - x^2y^2 - 2\,x^2y$. How can one show that $Q \geq 0$ on the unit disk? For example, by writing

$$Q(x, y) = (x^2 - y)^2 + (1 + x^2)\,(1 - x^2 - y^2)\ .$$

Indeed, $(x^2-y)^2$ and $1+x^2$ are SoS, and $1-x^2-y^2$ is non-negative on the unit disk. Is this method complete? No: there are non-negative polynomials that are not SoS. Is it correct? Obviously. Even better, it is "tractable", in a certain sense. Namely, while deciding "Is $P(\mathbf{x}) \geq 0$?" is NP-hard, the question "Is $P(\mathbf{x})$ a SoS" can be translated into a semi-definite program and then efficiently solved (at least up to arbitrary precision). This is the basic idea behind sum-of-square programs. In its full generality, however, it allows us much more. Namely, each line of a SoS program can be

1. A polynomial declaration, like "P is a polynomial of degree 6 in variables u, w, x, z". From the SoS program's point of view, the coefficients of P are *variables*. We call them *decision variables* since the SoS solver will decide what values to assign to them; the "true" variables u, w, x, z, etc. will be called *polynomial variables*.
2. A linear constraint in the decision variables.
3. A statement like "P is a Sum of Square polynomial".
4. A target function "minimize T", where T is a linear expression in the decision variables. Of course, there can be only one target function.

Thus, not only can we ask the SoS solver check that $P = (x^2+y)^2\,(1+z^2) - 4\,xyz$ is non-negative; we can also use it to find the maximum w for which $(x^2+y)^2\,(1+z^2) - wxyz$ is non-negative. There is no guarantee that it finds the maximum such value w; instead, it finds the maximum w for which $(x^2+y)^2\,(1+z^2) - wxyz$ is a SoS. Next, set $Q_w(x,y,z) = 1 - x^2y^2 - w\,x^2y$. We have seen above that Q_2 is non-negative on the unit disk. Can we determine the maximum w for which Q_w is non-negative on the unit disk? Again, the SoS framework offers no such guarantee, but we can write the following SoS program:

maximize	w
subject to	$Q_w = P_1 + P_2\,(1 - x^2 - y^2)$
	P_1, P_2 are SoS polynomials of degree d_1 and d_2, respectively.

Note that the constraint $Q_w = P_1 + P_2\,(1 - x^2 - y^2)$ is *linear in the coefficients of the polynomials*. Also, it is not clear a priori what degree d we should choose. As a heuristic, we could choose $d_1 = 4$ and $d_2 = 2$ to make sure that every summand on the right-hand side has degree 4, which is the degree of the left-hand side. If we are lucky, this returns the optimal w. In any case, the result will be a lower bound on the optimum: if P_1, P_2 are SoS and $Q_w = P_1 + P_2\,(1 - x^2 - y^2)$, then surely $Q_w \geq 0$ on the unit disk.

Let us now try to formulate our hunt for Cryptogenography upper bounds as a SoS program. Referring to Definition 1 and Theorem 2, we define f to be a polynomial of degree d in the variables $U = \{x_0, x_1, y_0, y_1\}$ with unknown coefficients. Point (a) and (b) of the definition are linear in the coefficients. How do we write (c) as a SoS constraint? First, define

$$g := -\left.\frac{\partial^2 f(\ell(t))}{\partial t^2}\right|_{t=0} .$$

This is a polynomial in the variables $V := U \cup \{r_0, r_1, s_0, s_1\}$. We could add the constraint "g is a SoS" into our program, but this requirement is way too strong. Indeed, g need be non-negative only for those values of V that constitute a player-line, i.e., that satisfy (2)–(6). We define $b_1 := x_0$, $b_2 := x_1$, $b_3 := y_0$, $b_4 := y_1$, $b_5 := s_0\,s_1$, $c_1 := x_0 + x_1 + y_0 + y_1 - 1$, and $c_2 := r_0 + r_1 + s_0 + s_1$, and $c_3 := y_0\,s_1 - y_1\,s_0$. Note that (2)–(6) are satisfied if and only if $b_1, \ldots, b_5 \geq 0$ and $c_1 = c_2 = c_3 = 0$. We add the following lines to our SoS program:

$$g = c_1 \cdot R_1 + c_2 \cdot R_2 + c_3 \cdot R_3 + \sum_{I \subseteq [5]} Q_I \cdot \prod_{i \in I} b_i \ , \tag{7}$$

each Q_I is a SoS polynomials of degree d_I ,

R_1, R_2, R_3 are arbitrary polynomials of degree d_R .

If g is as in (7), then $g \geq 0$ for all values of V constituting an Alice-line, and thus f is concave along all such lines. Obviously, we should add a similar constraint for Bob-lines. For efficiency reason we don't. Instead, we will exploit some inherent symmetries in the problem, as we explain in the next paragraph. Finally, we define the target function of our SoS program: "minimize $f(1/4, 1/4, 1/4, 1/4)$", which is a linear function in the coefficients of f.

3.1 Exploiting Symmetries

The Cryptogenography problem has two fundamental symmetries: we can switch the roles of Alice and Bob, and we can switch 0 and 1, all without changing the optimal value of a protocol for the uniform prior distribution. Suppose our generic admissible function f is a polynomial of degree d, namely $f(\mathbf{x}) = \sum_{\mathbf{a}\in\mathbb{N}_0^4:|\mathbf{a}|_1\leq d} w_{\mathbf{a}}\mathbf{x}^{\mathbf{a}}$, where $\mathbf{x} = (x_0, x_1, y_0, y_1)$. By symmetry, we can require that the $w_{a,b,c,d} = w_{b,a,d,c} = w_{c,d,a,b} = w_{d,c,b,a}$ for all $a, b, c, d \in \mathbb{N}_0$. Formally, for $\mathbf{a} = (a, b, c, d) \in \mathbb{N}_0^4$, let $\tilde{\mathbf{a}}$ denote the lexicographically first among (a, b, c, d), (b, a, d, c), (c, d, a, b), (d, c, b, a). Thus, we can write f as

$$f(\mathbf{x}) = \sum_{\mathbf{a}\in\mathbb{N}_0^4:|\mathbf{a}|_1\leq d} w_{\tilde{\mathbf{a}}}\mathbf{x}^{\mathbf{a}} \ . \tag{8}$$

If such a n f is concave along Alice-lines, it is also concave along Bob-lines, by symmetry. Thus, our SoS program only needs to contain a constraint for Alice-lines. As an additional bonus, formulating Point (a) and Point (b) of Definition 1 only takes two constraints rather than six. This basically describes our SoS program.

3.2 Obvious Optimizations

Note that (2) states $x_0 + x_1 + y_0 + y_1 = 1$ and (4) states that $r_0 + r_1 + s_0 + s_1 = 0$. Thus, instead of adding $c_1 \cdot R_1 + c_2 \cdot R_2$ to the right-hand side of (7), we can "by hand" substitute $x_1 = 1 - x_0 - y_0 - y_1$ and $r_1 = -r_0 - s_0 - s_1$. This reduces the number of polynomial variables from 8 to 6, which tremendously improves running time. The only downside is that our SoS program becomes a bit uglier. Putting everything together, here is the pseudocode of our SoS program:

$f(\mathbf{x}) := \sum_{\mathbf{a}\in\mathbb{N}_0^4:|\mathbf{a}|_1\leq d} w_{\tilde{\mathbf{a}}}\mathbf{x}^{\mathbf{a}}$.

minimize	$f(1/4, 1/4, 1/4, 1/4)$
subject to	$f(1, 0, 0, 0) \geq 0$
subject to	$f(1/2, 0, 1/2, 0) \geq 1/2$
	$g(x_0, x_1, y_0, y_1, r_0, r_1, s_0, s_1) := -\left.\frac{\partial^2 f(\ell(t))}{\partial t^2}\right\vert_{t=0}$
	$h(x_0, y_0, y_1, r_0, s_0, s_1) := g(x_0, 1 - x_0 - y_0 - y_1, y_0, y_1, r_0, -r_0 - s_0 - s_1, s_0, s_1)$
	$b_1 := x_0, b_2 := 1 - x_0 - y_0 - y_1, b_3 := y_0, b_4 := y_1, b_5 := s_0\, s_1, c := y_0\, s_1 - y_1\, s_0$
subject to	$h = \sum_{I\subseteq[5]} Q_I \cdot \prod_{i\in I} b_i + R \cdot c$
subject to	each Q_I is a SoS polynomial in x_0, y_0, y_1, r_0 of degree d_I,
	R is an arbitrary polynomial of degree d_R

It remains to specify the degree d_I of each SoS polynomial Q_I. We heuristically set d_I to $d - |I|$ or $d - |I| - 1$, whatever is even, and set $d_R := d - 2$. The intuition is that (a) the degree of the SoS polynomial Q_I must be even and (b) the degree of each summand on the right-hand side should be at most d, the degree of the left-hand side.

To implement the SoS program, we use the matlab package `yalmip` [9], since it turned out to run significantly faster on our problems than `sostools` [10]. In the "exploration phase" of this work we mostly used `sostools`, which might be a bit easier to work with for the non-experienced user. We wrote a `python` program that takes a degree d as input and outputs the `yalmip` code for our SoS program. In particular, it computes $f(\mathbf{x}) = \sum_{\mathbf{a}\in\mathbb{N}_0^4:|\mathbf{a}|_1\leq d} w_{\tilde{\mathbf{a}}}\mathbf{x}^{\mathbf{a}}$, taking care of all symmetries; the remaining `yalmip` code is the same for each d.

4 Handling Numerical Errors

We are solving a SoS program on a computer; this uses a numerical algorithm, so the result will contain some numerical errors. Inevitably so: SoS solvers are basically SDP solvers plus syntactic sugar, and some semi-definite programs only have irrational solutions.

In particular, the equation $h = \sum_{I\subseteq[5]} Q_I \cdot \prod_{i\in I} b_i + R\cdot c$ will only hold approximately. At least we can be sure that the Q_I on the right are SoS polynomials – we can ask the SoS solver to give us the SoS decomposition $p_1^2 + p_2^2 + \cdots + p_k^2$ and then simply define our Q_I to be this sum, and let $\tilde{h} := \sum_{I\subseteq[5]} Q_I \cdot \prod_{i\in I} b_i + R\cdot c$. However, $h = \tilde{h}$ will hold only approximately. Formally, $E := \tilde{h} - h$ is some non-zero polynomial with very small coefficients. By design, $\tilde{h} \geq 0$ for all values constituting an Alice-line; however, h might attain small negative values. If we had an "approximate version" of Theorem 2, stating that if f is "almost concave" along player-lines then it is "almost an upper bound", we would be done. Unfortunately, we do not have that, so we have to come up with a manual work-around. Let f be tentative upper bound function as found by the SoS solver. For some suitable $\epsilon > 0$, define

$$\hat{f}(\mathbf{x}) = f(\mathbf{x}) - \frac{\epsilon}{2}\,(x_0^2 + x_1^2 + y_0^2 + y_1^2) + \epsilon\ .$$

If f satisfies the boundary conditions up to error $\epsilon/2$, that is, if $f(1,0,0,0) \geq -\epsilon/2$ and $f(1/2,0,1/2,0) \geq 1/2 - \epsilon/2$, then $\hat{f}$ does indeed satisfy them. Next, we want to show that $\hat{f}$ satisfies the concavity constraints. By re-parametrizing the equation of a generic Alice-line (1), we can assume that $r_0^2 + r_1^2 + s_0^2 + s_1^2 = 1$. Put differently, one observes that g is homogenous of degree 2 in each of the variables r_0, r_1, s_0, s_1. So it is non-negative for all inputs in S if and only if it is for all inputs with $r_0^2 + r_1^2 + s_0^2 + s_1^2 = 1$. Consider the polynomials $\hat{g}$ and $\hat{h}$, defined analogous to g and h, but starting with $\hat{f}$ instead of f. A brief calculation shows that $\hat{g} = g + \epsilon\cdot(r_0^2 + r_1^2 + s_0^2 + s_1^2) = g + \epsilon$ and thus $\hat{h} = h + \epsilon$. Thus, on any input $(\mathbf{x},\mathbf{r}) \in S$ with $\|\mathbf{r}\|_2 = 1$, we have $h(\mathbf{x},\mathbf{r}) = \tilde{h}(\mathbf{x},\mathbf{r}) - E(\mathbf{x},\mathbf{r}) + \epsilon \geq \epsilon - E(\mathbf{x},\mathbf{r})$. We give a very crude upper bound on $E(\mathbf{x},\mathbf{r})$: since all input variables are at most 1 in absolute value, $E(\mathbf{x},\mathbf{r})$ is at most the sum of all coefficients. Let us denote this quantity by $\|E\|_1$. Thus, as long as $\epsilon \geq \|E\|_1$, we can be sure that $\hat{f}$ is indeed admissible, and thus an upper bound. That is, $\mathrm{val}(1/4,1/4,1/4,1/4) \leq \hat{f}(1/4,1/4,1/4,1/4) \leq f(1/4,1/4,1/4,1/4) + \|E\|_1$. The table in Section 1.2 sums up the results for degree $2,4,6,8$. The column labeled "numerical error" is our upper bound $\|E\|_1$.

One could attempt to further reduce the number of variables. Indeed, equality (5) states $y_0\, s_1 = y_1\, s_0$. Using this, we could scale (1) to ensure that $s_0 = y_0$ and $s_1 = y_1$. This reduces the number of variables from six to four, and vastly reduces the running time of `yalmip`. However, we cannot simultaneously ensure that $\|r\|_2 = 1$ and thus do not know how to handle the resulting numerical error. It would still be interesting to see what this gives for degree 10 and 12, but it would require a leap of faith to conclude that the bound thus computed really holds.

5 Open Questions

Is there a better way to handle numerical errors? If so, then maybe one *can* remove the number of variables to four and explore higher degrees. As for theoretical aspects, can one prove that as $d \to \infty$, the optimal value of the SoS program converges to the actual optimum of the cryptogenography problem? It is known (see for example Lasserre [7]) that any non-negative $f : [-1, 1]^m \to \mathbb{R}$ can be arbitrarily well approximated by SoS polynomials. However, we don't require the polynomial g to be non-negative everywhere; only non-negative on a certain set. It is not clear to me whether the approximation result transfers to this setting.

6 Source Code and Data

Since already for degree four, the number of monomials in f and the polynomials Q_I are too large to be comfortably printed in an article, we created the webpage

`http://basics.sjtu.edu.cn/~dominik/sos-cryptogenography/`

to which we uploaded all source code and all output data that enables the reader to verify our results without having to run `yalmip` or any SoS solver.

References

1. Mark Braverman, Ankit Garg, Denis Pankratov, and Omri Weinstein. From information to exact communication. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 151–160. ACM, 2013. doi:10.1145/2488608.2488628.
2. Joshua Brody, Sune K. Jakobsen, Dominik Scheder, and Peter Winkler. Cryptogenography. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 13–22. ACM, 2014. doi:10.1145/2554797.2554800.
3. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006.
4. Benjamin Doerr and Marvin Künnemann. Improved Protocols and Hardness Results for the Two-Player Cryptogenography Problem. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 150:1–150:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.150.
5. Sune K. Jakobsen. Information Theoretical Cryptogenography. *J. Cryptology*, 30(4):1067–1115, 2017. doi:10.1007/s00145-016-9242-8.
6. Sune K. Jakobsen and Claudio Orlandi. How To Bootstrap Anonymous Communication. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, pages 333–344, New York, NY, USA, 2016. ACM. doi:10.1145/2840728.2840743.
7. Jean B. Lasserre. A Sum of Squares Approximation of Nonnegative Polynomials. *SIAM J. on Optimization*, 16(3):751–765, March 2006. doi:10.1137/04061413X.
8. Monique Laurent. Sums of Squares, Moment Matrices and Optimization Over Polynomials. In Mihai Putinar and Seth Sullivant, editors, *Emerging Applications of Algebraic Geometry*, pages 157–270. Springer New York, New York, NY, 2009. doi:10.1007/978-0-387-09686-5_7.
9. J. Löfberg. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.
10. A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, and P. A. Parrilo. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*, 2013. Available from `http://www.eng.ox.ac.uk/control/sostools`, `http://www.cds.caltech.edu/sostools` and `http://www.mit.edu/~parrilo/sostools`. arXiv:1310.4716.

11 Antonis Papachristodoulou, James Anderson, Giorgio Valmorbida, Stephen Prajna, Pete Seiler, and Pablo A. Parrilo. SOSTOOLS version 3.00 sum of squares optimization toolbox for MATLAB. *CoRR*, abs/1310.4716, 2013. arXiv:1310.4716.

A The yalmip Code for Degree 4

```
clear
yalmip('clear')
tic
echo on

%%% begin python generated code
degree = 4;
sdpvar a b c d r0 s0 s1 t  w0000 w1000 w2000 w1100 w1010 w1001 w3000
w2100 w2010 w2001 w1110 w4000 w3100
 w3010 w3001 w2200 w2110 w2101 w2020 w2011 w2002 w1111;
fvars = [w0000; w1000; w2000; w1100; w1010; w1001; w3000; w2100;
w2010; w2001; w1110; w4000; w3100; w3010;
w3001; w2200; w2110; w2101; w2020; w2011; w2002; w1111];
decisionvars = fvars;
f = w0000*1 + w1000*a + w1000*b + w1000*c + w1000*d + w2000*a^2 + w1100*a*b +
w1010*a*c + w1001*a*d + w2000*b^2 + w1001*b*c + w1010*b*d + w2000*c^2 + w1100*c*d +
w2000*d^2 + w3000*a^3 + w2100*a^2*b + w2010*a^2*c + w2001*a^2*d + w2100*a*b^2 +
w1110*a*b*c + w1110*a*b*d + w2010*a*c^2 + w1110*a*c*d + w2001*a*d^2 + w3000*b^3 +
w2001*b^2*c + w2010*b^2*d + w2001*b*c^2 + w1110*b*c*d + w2010*b*d^2 + w3000*c^3 +
w2100*c^2*d + w2100*c*d^2 + w3000*d^3 + w4000*a^4 + w3100*a^3*b +  w3010*a^3*c +
w3001*a^3*d + w2200*a^2*b^2 +  w2110*a^2*b*c + w2101*a^2*b*d + w2020*a^2*c^2 +
w2011*a^2*c*d +  w2002*a^2*d^2 + w3100*a*b^3 + w2101*a*b^2*c  + w2110*a*b^2*d +
w2011*a*b*c^2 + w1111*a*b*c*d + w2011*a*b*d^2 +  w3010*a*c^3 + w2110*a*c^2*d +
w2101*a*c*d^2 + w3001*a*d^3 + w4000*b^4 + w3001*b^3*c +  w3010*b^3*d +
w2002*b^2*c^2 + w2011*b^2*c*d + w2020*b^2*d^2 + w3001*b*c^3 + w2101*b*c^2*d +
w2110*b*c*d^2 +  w3010*b*d^3 + w4000*c^4 + w3100*c^3*d + w2200*c^2*d^2 +
w3100*c*d^3 + w4000*d^4;
%%% end python generated code


B = 1-a-c-d;
R1 = -r0-s0-s1;
f_replace = replace(f, [a,b,c,d], [a + t*r0, B + t*R1, c + t*s0, d + t*s1])
h = -replace(jacobian(jacobian(f_replace ,t), t), [t], [0])


corner = replace(f, [a,b,c,d], [1,0,0,0]);
edge = replace(f, [a,b,c,d], [1/2,0,1/2,0]);

Constraintsp0 = [corner == 0, edge == 1/2];
Constraintsp = [];

%%% concavity constraint for Alice:

%% Our set S of feasible inputs is defined by five polynomial inequalities
%% and one equality
%% The five inequalities are a, B, c, d, s0*s1 >= 0
%% Each subset of [5] gives us a 'region polynomial', like a*B*s0*s1
```

```
%% For technical reasons we only list the products over non-empty subsets:

region_vector = [a, B, c, d, a*B, a*c, a*d, B*c, B*d, c*d, a*B*c, a*B*d, a*c*d,
B*c*d, a*B*c*d, s0*s1, a*s0*s1, B*s0*s1, c*s0*s1, d*s0*s1, a*B*s0*s1,
a*c*s0*s1, a*d*s0*s1, B*c*s0*s1, B*d*s0*s1, c*d*s0*s1, a*B*c*s0*s1,
a*B*d*s0*s1, a*c*d*s0*s1, B*c*d*s0*s1, a*B*c*d*s0*s1];


%% This is simply a list containing the degrees of the polynomials just defined

deg_of_boundaries = [1,1,1,1,2,2,2,2,2,2,3,3,3,3,4,2,3,3,3,3,4,4,4,4,4,4,5,5,5,5,6];

%% We now define our polynomails Q_I for each (non-empty) subset I of [5]
%% Each Q_I is in the variables a,c,d,s0, s1, r0:

polyvars = [a,c,d, s0, s1, r0];
boundary_polys = [];
for i=1:length(region_vector)

    % Creating the polynomial Q_I
    % remember: the degree of Q_I * prod_{i in I} b_i should be at most <degree>,
    % and the degree of Q_I must be even

    degree_here = max(0,2*floor((degree - deg_of_boundaries(i))/2));

    % We ask yalmip to create a new polynomial Q_I
    [Q_I, coeff] = polynomial(polyvars, degree_here);
    boundary_polys = [boundary_polys; Q_I];

    % this Q_I must be SoS, so add a SoS constraint
    Constraintsp = [Constraintsp, sos(Q_I)];

    % tell yalmip that we created a new bunch of decision variables:
    % the coefficients of Q_I
    decisionvars = [decisionvars; coeff];
end

% In the paper we had the (not necessarily SoS) polynomial R
% which enters the SoS program as R*(y0*s1 - y1*s0)
% Here, we use c,d instead of y0, y1 and, for clarity
% 'alice_line_poly' instead of 'R'

[alice_line_poly, alice_line_poly_coeff] = polynomial(polyvars, degree-2);
decisionvars = [decisionvars; alice_line_poly_coeff];

[normalization_poly, normalization_poly_coeff] = polynomial(polyvars, degree-2);
decisionvars = [decisionvars; normalization_poly_coeff];


Constraints = [sos(h - region_vector * boundary_polys ...
    - (c*s1 - d*s0) * alice_line_poly ...
    - (1 - r0^2 - R1^2 - s0^2 - s1^2) * normalization_poly), Constraintsp];
Constraints = [Constraintsp0, Constraints];

target = replace(f, [a,b,c,d], [1/4,1/4,1/4,1/4]);
```

```
ops = sdpsettings('solver','sedumi','sedumi.eps',1e-14);

%% finally solving the SoS program

solvesos(Constraints, target, [ops] , decisionvars);



%% assembling the right-hand side of h, i.e., what we call \tilde{h} in the paper

sdpvar Q0_sos_polys Qi_sos_poly sum_poly error_poly error_poly_sym

sum_poly = 0;

Q0_sos_polys = sosd(Constraints(3));
for i=1:length(Q0_sos_polys)
    sum_poly = sum_poly + Q0_sos_polys(i)^2;
end

for k=1:length(region_vector)
    Qi_sos_polys = sosd(Constraints(k+3));
    for i=1:length(Qi_sos_polys)
        sum_poly = sum_poly + region_vector(k)*(Qi_sos_polys(i)^2);
    end
end

sum_poly = sum_poly ...
    + (c*s1 - d*s0) * replace(alice_line_poly, decisionvars, value(decisionvars))
    + (1 - r0^2 - R1^2 - s0^2 - s1^2) *
    replace(normalization_poly, decisionvars, value(decisionvars));

error_poly = sum_poly - replace(h, decisionvars, value(decisionvars));

error_poly_sym = str2sym( sdisplay(error_poly) );

syms vec_error vec_sum;

vec_error = coeffs(error_poly_sym);

total_error = 0;
for i=1:length(vec_error)
    total_error = total_error + abs(vec_error(i));
end

sdisplay(replace(target,decisionvars,value(decisionvars)))
total_error

toc
```