Thrifty Zero-Knowledge
When Linear Programming Meets Cryptography

Simon Cogliani, Houda Ferradi, Rémi Géraud, and David Naccache

École normale supérieure, Information Security Group,
45 rue d'Ulm, F-75230 Paris CEDEX 05, France
given_name.family_name@ens.fr

Abstract. We introduce "thrifty" zero-knowledge protocols, or TZK. These protocols are constructed by introducing a bias in the challenge send by the prover. This bias is chosen so as to maximize the security versus effort trade-off. We illustrate the benefits of this approach on several well-known zero-knowledge protocols.

1 Introduction

Since their discovery, zero-knowledge proofs (ZKPs) [3, 10] have found many applications and have become of central interest in cryptology. ZKPs enable a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that some mathematical statement is valid, in such a way that no knowledge but the statement's validity is communicated to $\mathcal{V}$. The absence of information leakage is formalized by the existence of a simulator $\mathcal{S}$, whose output is indistinguishable from the recording (trace) of the interaction between $\mathcal{P}$ and $\mathcal{V}$.

Thanks to this indistinguishability, an eavesdropper $\mathcal{A}$ cannot tell whether she taps a real conversation or the monologue of $\mathcal{S}$. $\mathcal{P}$ and $\mathcal{V}$, however, interact with each other and thus know that the conversation is real.

It may however happen, by sheer luck, that $\mathcal{A}$ succeeds in responding correctly to a challenge without knowing $\mathcal{P}$'s secret. ZKPs are designed so that such a situation is expected to happen only with negligible probability: Repeating the protocol renders the cheating probability exponentially small *if* the challenge at each protocol round is random. Otherwise, $\mathcal{A}$ may repeat her successful commitments while hoping to be served with the same challenges.

Classically, the protocol is regarded as ideal when the challenge distribution is *uniform* over a large set (for efficiency reasons, the cardinality of this set rarely exceeds 2^{128}). Uniformity, however, has its drawbacks: all challenges are not computationally equal, and some challenges may prove harder than others to respond to.

This paper explores the effect of biasing the challenge distribution. Warping this distribution unavoidably sacrifices security, but it appears that the resulting efficiency gains balance this loss in a number of ZKPs. Finding the optimal distribution brings out interesting optimization problems which happen to be solvable exactly for a variety of protocols and variants. We apply this idea to

improve on four classical ZK identification protocols that rely on very different assumptions: RSA-based Fiat-Shamir [8], SD-based identification [18], PKP-based identification [17], and PPP-based indentification [16].

2 Preliminaries

2.1 Three-Round Zero-Knowledge Protocols

A Σ-protocol [4, 9, 11] is a generic 3-step interactive protocol, whereby a prover $\mathcal{P}$ tries to convince a verifier $\mathcal{V}$ that $\mathcal{P}$ knows a proof that some statement is true — without revealing anything to $\mathcal{V}$ beyond this assertion. The three phases of a Σ-protocol are illustrated by Figure 1.

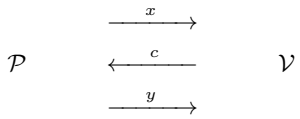

Fig. 1. Generic Σ-protocol.

Namely,

- $\mathcal{P}$ sends a *commitment* x to $\mathcal{V}$
- $\mathcal{V}$ replies with a *challenge* c;
- $\mathcal{P}$ provides a *response* y.

Upon completion, $\mathcal{V}$ may accept or reject $\mathcal{P}$, depending on whether $\mathcal{P}$'s response is satisfactory. In practice, the protocol will be repeated several times until $\mathcal{V}$ is satisfied.

An eavesdropper $\mathcal{A}$ should not be able to learn anything from the conversation between $\mathcal{P}$ and $\mathcal{V}$. This security notion is formalized by the existence of a simulator $\mathcal{S}$, whose output is indistinguishable from the interaction (or "trace") T between $\mathcal{P}$ and $\mathcal{V}$. Different types of zero-knowledge protocols exist, that correspond to different indistinguishability notions.

In *computational* zero-knowledge, $\mathcal{S}$'s output distribution is computationally indistinguishable from T, whereas in *statistical* zero-knowledge, $\mathcal{S}$'s output distribution must be statistically close to the distribution governing T: Thus even a computationally unbounded verifier learns nothing from T. The strongest notion of *unconditional* zero-knowledge requires that $\mathcal{A}$ cannot distinguish $\mathcal{S}$'s output from T, even if $\mathcal{A}$ is given access to both unbounded computational resources and $\mathcal{P}$'s private keys. The Fiat-Shamir protocol [8] is an example of unconditional ZKP.

Definition 1 (Statistical Indistinguishability). *The statistical difference between random variables X and Y taking values in $\mathcal{Z}$ is defined as:*

$$\begin{aligned}\Delta(X,Y) &:= \max_{Z\subset\mathcal{Z}} |\Pr(X\in Z) - \Pr(Y\in Z)| \\ &= 1 - \sum_{z\in\mathcal{Z}} \min\{\Pr(X=z), \Pr(Y=z)\}\end{aligned}$$

We say that X and Y are statistically indistinguishable *if $\Delta(X,Y)$ is negligible.*

Finally, we expect $\mathcal{P}$ to eventually convince $\mathcal{V}$, and that $\mathcal{V}$ should only be convinced by such a $\mathcal{P}$ (with overwhelming probability). All in all, we have the following definition:

Definition 2 (Σ-protocol). *A Σ-protocol is a three-round protocol that furthermore satisfies three properties:*

- Completeness*: given an input v and a witness w such that vRw, $\mathcal{P}$ is always able to convince $\mathcal{V}$.*
- Zero-Knowledge*: there exists a probabilistic polynomial-time simulator $\mathcal{S}$ which, given (v,c), outputs triples (x,c,y) that follow a distribution indistinguishable from a valid conversation between $\mathcal{P}$ and $\mathcal{V}$.*
- Special Soundness*: given two accepting conversations for the same input v, and the same commitment x, but with different challenges $c_1 \neq c_2$, there exists a probabilistic polynomial-time algorithm $\mathcal{E}$ called extractor that computes a witness $w = \mathcal{E}(c_1, c_2, v, x)$ such that vRw.*

2.2 Security Efficiency

During a Σ-protocol, $\mathcal{P}$ processes c to return the response $y(x,c)$. The amount of computation $W(x,c)$ required for doing so depends on x, c, and on the challenge size, denoted k. Longer challenges — hence higher security levels — would usually claim more computations.

Definition 3 (Security Level). *Let $\mathcal{P}\leftrightarrow\mathcal{V}$ be a Σ-protocol, the* security level *$S(\mathcal{P}\leftrightarrow\mathcal{V})$: is defined as the challenge min-entropy*

$$S(\mathcal{P}\leftrightarrow\mathcal{V}) := -\min_{c} \log \Pr(c)$$

This security definition assumes that $\mathcal{A}$'s most rational attack strategy is to focus her efforts on the most probable challenge. From a defender's perspective, verifiers achieve the highest possible security level by sampling challenges from a uniform distribution.

Definition 4 (Work Factor). *Let $\mathcal{P}\leftrightarrow\mathcal{V}$ be a Σ-protocol, the* average work factor *$W(\mathcal{P}\leftrightarrow\mathcal{V})$ is defined as the expected value of $W(x,c)$:*

$$W(\mathcal{P}\leftrightarrow\mathcal{V}) := \mathbb{E}_{x,c}\left[W(x,c)\right]$$

Definition 5 (Security Efficiency). *Let $\mathcal{P} \leftrightarrow \mathcal{V}$ be a Σ-protocol, the* security efficiency *of $\mathcal{P} \leftrightarrow \mathcal{V}$, denoted $E(\mathcal{P} \leftrightarrow \mathcal{V})$, is defined as the ratio between $S(\mathcal{P} \leftrightarrow \mathcal{V})$ and $W(\mathcal{P} \leftrightarrow \mathcal{V})$:*

$$E(\mathcal{P} \leftrightarrow \mathcal{V}) := \frac{S(\mathcal{P} \leftrightarrow \mathcal{V})}{W(\mathcal{P} \leftrightarrow \mathcal{V})}$$

Informally, $E(\mathcal{P} \leftrightarrow \mathcal{V})$ represents[1] the average number of security bits per mathematical operation.

2.3 Linear Programming

Linear programming (LP) [2, 5–7] problems appear when a linear objective function must be optimized under linear equality and inequality constraints. These constraints define a convex polytope. General linear programming problems can be expressed in canonical form as:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} \leq \boldsymbol{b} \\ \text{and} & \boldsymbol{x} \geq 0 \end{array}$$

where $\boldsymbol{x}$ represents the vector of variables (to be determined), $\boldsymbol{c}$ and $\boldsymbol{b}$ are vectors of (known) coefficients and A is a (known) matrix of coefficients.

Linear programming is common in optimization problems and ubiquitous in logistics, operational research, and economics. Interestingly, linear programming has almost never surfaced in cryptography, save a few occasional appearances in error correcting codes [1], or under the avatar of its NP-hard variant, integer programming [14].

Every linear problem can be written in so-called "standard form" where the constraints are all inequalities and all variables are non-negative, by introducing additional variables ("slack variables") if needed. Not all linear programming problems can be solved: The problem might be unbounded (there is no maximum) or infeasible (no solution satisfies the constraints, *i.e.* the polytope is empty).

Many algorithms are known to solve LP instances, on the forefront Dantzig's Simplex algorithm [5]. The Simplex algorithm solves an LP problem by first finding a solution compatible with the constraints at some polytope vertex, and then walking along a path on the polytope's edges to vertices with non-decreasing values of the objective function. When an optimum is found the algorithm terminates — in practice this algorithm has usually good performance but has poor worst-case behavior: There are LP problems for which the Simplex method takes a number of steps exponential in the problem size to terminate [6, 15].

Since the 1950's, more efficient algorithms have been proposed called "interior point" methods (as opposed to the Simplex which evolves along the polytope's vertices). In particular, these algorithms demonstrated the polynomial-time solvability of linear programs [12]. Following this line of research, approximate

[1] *i.e.* is proportional to

solutions to LP problems can be found using very efficient (near linear-time) algorithms [13, 19].

In this work we assume that some (approximate) LP solver is available. Efficiency is not an issue, since this solver is only used once, when the ZKP is designed

3 Optimizing $E(\mathcal{P} \leftrightarrow \mathcal{V})$

The new idea consists in assigning *different* probabilities to different c values, depending on how much it costs to generate their corresponding y values, while achieving a given security level. The intuition is that by choosing a certain distribution of challenges, we may hope to reduce $\mathcal{P}$'s total amount of effort, but this also reduces security. As we show, finding the best trade-off is equivalent to solving an LP problem.

Consider a set Γ of symbols, and a cost function $\eta : \Gamma \to \mathbb{N}$. Denote by $p_j := \Pr(i \mid i \in \Gamma_j)$ the probability that a symbol i is emitted, given that i has cost j. We wish to find this probability distribution.

Let Γ_j denote all symbols having cost j, *i.e.* such that $\eta(i) = j$. Let γ_j be the cardinality of Γ_j. The expected cost for a given choice of emission probabilities $\{p_j\}$ is

$$W = \mathbb{E}[\eta] = \sum_{i \in \Gamma} \eta(i) \Pr(i) = \sum_j j \times \gamma_j \times p_j$$

W is easy to evaluate provided we can estimate the amount of work associated with each challenge isocost class Γ_j. The condition that probabilities sum to one is expressed as:

$$1 = \sum_{i \in \Gamma} \Pr(i) = \sum_j \gamma_j p_j$$

Finally, security is determined by the min-entropy

$$S = -\log_2 \max_i \Pr(i) = -\log_2 \max_j p_j$$

Let $\epsilon = 2^{-S}$, so that $p_j \leq \epsilon$ for all j. The resulting security efficiency is $E = S/W = (-\log_2 \epsilon)/W$.

We wish to maximize E, which leads to the following constrained optimization problem:

$$\text{Given } \{\gamma_j\} \text{ and } \epsilon, \begin{cases} \text{minimize} & W = \sum_j j p_j \gamma_j \\ \text{subject to} & 0 \leq p_j \leq \epsilon \\ & \sum_j \gamma_j p_j = 1 \end{cases} \tag{1}$$

This is a linear programming problem [5–7], that can be put in canonical form by introducing slack variables $q_j = \epsilon - p_j$ and turning the inequality constraints into equalities $p_j + q_j = \epsilon$. The solution, if it exists, therefore lies on the boundary of the polytope defined by these constraints.

Note that a necessary condition for an optimal solution to exist is that $\epsilon \geq 1/\sum_j \gamma_j$, which corresponds to the choice of the uniform distribution.

Exact solutions to Equation (1) can be found using the techniques mentioned in Section 2.3.

We call such optimized ZKP versions "thrifty ZKPs". Note that the zero-knowledge property is not impacted, as it is trivial to construct a biaised simulator.

4 Thrifty Zero-Knowledge Protocols

The methodology described in Section 3 can be applied to any ZK protocol, provided that we can evaluate the work factor associated with each challenge class. As an illustration we analyse thrifty variants of classical ZKPs: Fiat-Shamir (FS, [8]), Syndrome Decoding (SD, [18]), Permuted Kernels Problem (PKP, [17]), and Permuted Perceptrons Problem (PPP, [16]).

4.1 Thrifty Fiat-Shamir

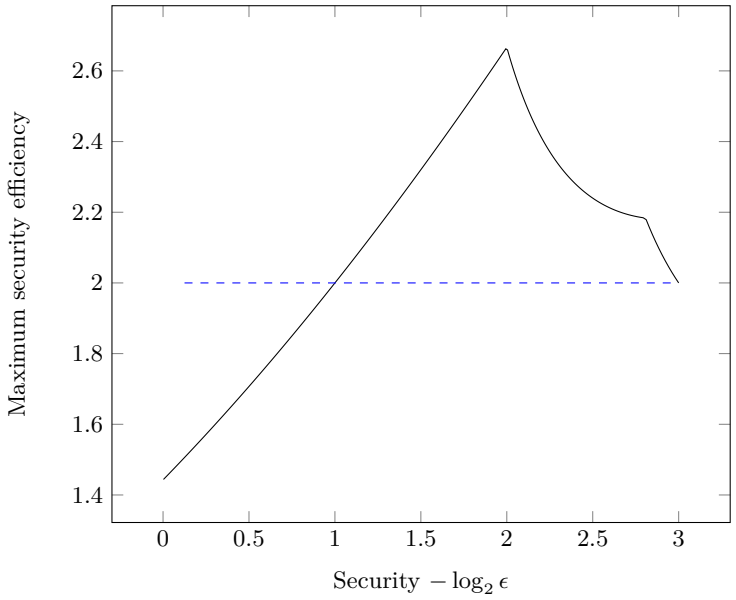

Fig. 2. Security efficiency for biased Fiat-Shamir with $n = 3$, as a function of ϵ. Standard Fiat-Shamir security efficiency corresponds to the dashed line.

In the case of Fiat-Shamir [8] (see [8]), response to a challenge c claims a number of multiplications proportional to c's Hamming weight. We have $k = n$-bit

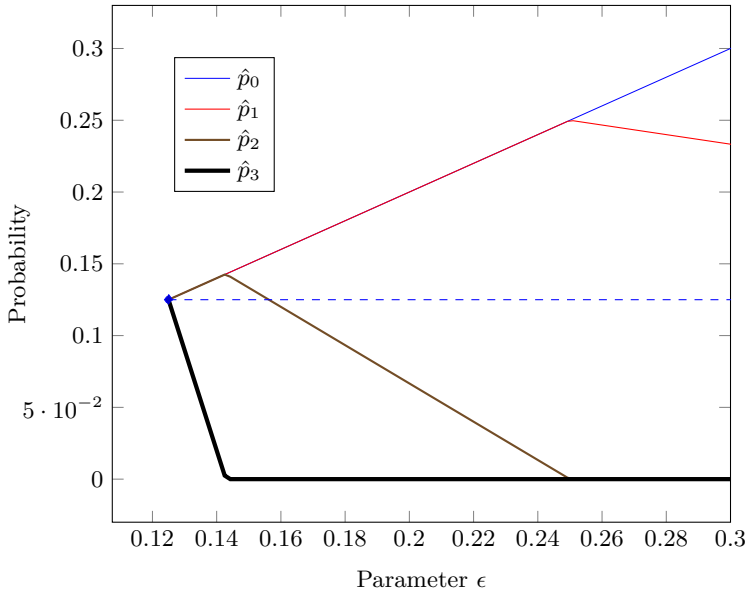

Fig. 3. Fiat-Shamir ($k = n = 4$) optimal probability distribution for challenges in group $j = 0, \ldots, 3$, as a function of ϵ. Branching happens at $\epsilon = 1/7$ and $\epsilon = 1/4$. Dashed line corresponds to the standard Fiat-Shamir distribution.

long challenges. Here γ_j is the number of n-bit challenges having Hamming weight j, namely

$$\gamma_j = \binom{n}{j}$$

Note that the lowest value of ϵ for which a solution to Equation (1) exists is 2^{-n}, in which case $p_j = \epsilon$ is the uniform distribution, and $W = n/2$. Hence the original Fiat-Shamir always has $E = 2$.

Example 1. Let $n = 4$. In that case Equation (1) becomes the following problem:

$$\text{Given } \epsilon, \begin{cases} \text{minimize} & W = 3p_1 + 6p_2 + 3p_3 \\ \text{subject to} & 0 \le p_0, p_1, p_2, p_3 \le \epsilon \\ & p_0 + 3p_1 + 3p_2 + p_3 = 1 \end{cases}$$

Security efficiency is $(-\log_2 \epsilon)/W$. Note that the original Fiat-Shamir protocol has $W = 3/2$ and security $S = 3$ bits, hence a security efficiency of $E = 2$, as pointed out previously.

Let for instance $\epsilon = 1/7$, for which the solution can be expressed simply as $p_0 = p_1 = p_2 = \epsilon$, and $p_3 = 1 - 7\epsilon$, yielding an effort

$$W = 9\epsilon + 3(1 - 7\epsilon) = 3(1 - 4\epsilon)$$

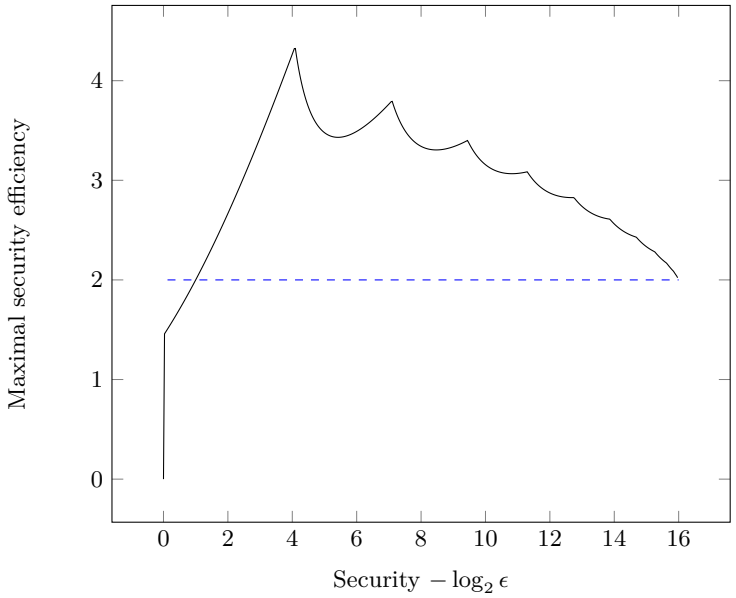

Fig. 4. Maximal security efficiency $\hat{E}$ for biased Fiat-Shamir with $n = 16$, as a function of security $-\log \epsilon$. Standard Fiat-Shamir security efficiency corresponds to the dashed line.

Therefore the corresponding security efficiency is $\frac{-\log_2 \epsilon}{3(1-4\epsilon)}$, which at $\epsilon = 1/7$ equals $7 \log_2 7/9 \simeq 2.18$. This is a 10% improvement over a standard Fiat-Shamir.

Remark 1. We can compute the optimal distribution for any value of $\epsilon \geq 1/8$, *i.e.* choose the p_is that yields the maximum security efficiency $\hat{E}(\epsilon)$. The result of this computation is given in Figure 2. Corresponding optimal probabilities $\hat{p}_i$ are given in Figure 3.

Remark 2. Figure 2 shows that $\hat{E}$ is not a continuously differentiable function of ϵ. The two singular points correspond to $\epsilon = 1/7$ and $\epsilon = 1/4$. These singular points correspond to optimal strategy changes: when ϵ gets large enough, it becomes interesting to reduce the probability of increasingly many symbols. This is readily observed on Figure 3 which displays the optimal probability distribution of each symbol group as a function of ϵ.

Example 2. Solving Equation (1) for Fiat-Shamir with $n = 16$ gives Figure 4 which exhibits the same features as Figure 2, with more singular points positioned at $\epsilon = 2^{-4}, 2^{-7}, 2^{-9}$, etc.

Table 1. Challenge effort distribution for SD [18], with a 16×16 parity matrix H, over 10^4 runs.

Challenge	Operations by prover	Time	Optimal p_i
0	Return y and σ	0 s ± 0.01	0.5
1	Compute $y \oplus s$	747.7 s ± 2	0.1
2	Compute $y \cdot \sigma$ and $s \cdot \sigma$	181.22 s ± 2	0.4

Table 2. Challenge effort distribution for PKP [17], over 10^7 runs.

Challenge	Operations by prover	Time	Optimal p_i
0	Compute W	390 s ± 2	0.6
1	Compute W and $\pi(\sigma)$	403 s ± 2	0.4

Table 3. Challenge effort distribution for PPP!, over 10^6 runs.

Challenge	Operations by prover	Time	Optimal p_i
0	Return P, Q, W	0.206 s ± 0.05	0.5
1	Compute $W + Q^{-1}V$	6.06 s ± 0.05	0.17
2	Compute $Q(P(A))$ and $Q^{-1}V$	21.13 s ± 0.5	0.1
3	Compute $Q^{-1}V$	4.36 s ± 0.05	0.23

4.2 Thrifty SD, PKP and PPP

The authors implemented[2] the SD, PKP and PPP protocols, and timed their operation as a function of the challenge class. Only the relative time taken by each class is relevant, and can be used as a measure of $\mathcal{W}$. The methodology of Section 3 is then used to compute the optimal probability distributions and construct the thrifty variant of these protocols.

The result of these measurements[3] is summarized in Tables 1 to 3. For details about the protocols we refer the reader to the original descriptions. All in all, we achieve up to 20% gain in security efficiency using our approach, on both PPP, PKP and SD.

References

1. Bierbrauer, J., Gopalakrishnan, K., Stinson, D.R.: Bounds for resilient functions and orthogonal arrays. In: Desmedt, Y. (ed.) Advances in Cryptology – CRYPTO'94.

[2] Python source code is available upon request.

[3] Experiments were performed on a Intel Core i7-4712HQ CPU at 2.30 GHz, running Linux 3.13.0, Python 2.7.6, `numpy` 1.9.3, and `sympy` 0.7.6.1.

Lecture Notes in Computer Science, vol. 839, pp. 247–256. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 21–25, 1994)

2. Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
3. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. 37(2), 156–189 (1988)
4. Damgård, I.: On Σ Protocols (2010), http://www.cs.au.dk/~ivan/Sigma.pdf
5. Dantzig, G.B.: Maximization of a linear function of variables subject to linear inequalities. Activity Analysis of Production and Allocation (1951)
6. Dantzig, G.B., Thapa, M.N.: Linear programming 1: Introduction. Springer Science & Business Media (2006)
7. Dantzig, G.B., Thapa, M.N.: Linear programming 2: Theory and extensions. Springer Science & Business Media (2006)
8. Feige, U., Fiat, A., Shamir, A.: Zero-knowledge proofs of identity. J. Cryptology 1(2), 77–94 (1988)
9. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. J. ACM 38(3), 691–729 (1991)
10. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. 18(1), 186–208 (1989)
11. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols - Techniques and Constructions. Information Security and Cryptography, Springer (2010)
12. Karmarkar, N.: A new polynomial-time algorithm for linear programming. In: Proceedings of the sixteenth annual ACM symposium on Theory of computing. pp. 302–311. ACM (1984)
13. Koufogiannakis, C., Young, N.E.: Beating simplex for fractional packing and covering linear programs. CoRR abs/0801.1987 (2008), http://arxiv.org/abs/0801.1987
14. Lenstra, H.: Integer programming and cryptography. The Mathematical Intelligencer 6(3), 14–21 (1984)
15. Murty, K.G.: Linear programming (1983)
16. Pointcheval, D.: A new identification scheme based on the perceptrons problem. In: Guillou, L.C., Quisquater, J.J. (eds.) Advances in Cryptology – EUROCRYPT'95. Lecture Notes in Computer Science, vol. 921, pp. 319–328. Springer, Heidelberg, Germany, Saint-Malo, France (May 21–25, 1995)
17. Shamir, A.: An efficient identification scheme based on permuted kernels (extended abstract) (rump session). In: Brassard, G. (ed.) Advances in Cryptology – CRYPTO'89. Lecture Notes in Computer Science, vol. 435, pp. 606–609. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990)
18. Stern, J.: A new identification scheme based on syndrome decoding. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO'93. Lecture Notes in Computer Science, vol. 773, pp. 13–21. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994)
19. Zhu, Z.A., Orecchia, L.: Using optimization to break the epsilon barrier: A faster and simpler width-independent algorithm for solving positive linear programs in parallel. CoRR abs/1407.1925 (2014), http://arxiv.org/abs/1407.1925

A Source Code

The following implementation uses Python 2.7 and the CVXOPT library[4] to solve the constrained optimization problem of Equation (1). Here the γ_j are computed for Fiat-Shamir, but could easily be adapted to other settings.

```
from cvxopt import matrix, solvers
from fractions import Fraction
import math

mul = lambda x,y: x*y

# Binomial coefficient \binom{n}{k}
def binom(n,k):
  return int(reduce(mul,(Fraction(n-i,i+1) for i in range(k)),1))

# Populations \gamma_k (for Fiat-Shamir)
def get_coeffsp(n):
  return [binom(n,k+1) for k in range(n)]

# Work coefficients k * \gamma_k (for Fiat-Shamir)
def get_coeffsw(n):
  r = get_coeffsp(n)
  return [(i+1)*c for i,c in enumerate(r)]

# Solve optimization problem for given n and epsilon
def solve_lp(epsilon, n):

  coeffsp = map(float, get_coeffsp(n))
  coeffsw = map(float, get_coeffsw(n))

  # Put the problem in canonical form, i.e.
  # construct matrix A and vectors b, c
  # such that the problem is in the form Ax + b <= c
  A = []
  for i in range(n):
    A += [[0.]*i + [1.] + [0.]*(n-i-1)]
  A += [map(lambda y:-y, coeffsp)]
  for i in range(n):
    A += [[0.]*i + [-1.] + [0.]*(n-i-1)]
  A += [coeffsp]
  A = matrix(A).trans()
  b = matrix([epsilon] * n + [epsilon-1.] + [0.] * n + [1.])
  c = matrix(coeffsw)

  # Solve the linear programming problem
  sol = solvers.lp(c, A, b)

  # Extract solution and append p0
  p0 = 1 - sum(i*w for i, w in zip(sol['x'], coeffsp))
  pi = [p0] + [i for i in sol['x']]

  # Compute total work (for Fiat-Shamir)
  w = sum(i * w for i, w in zip(sol['x'], coeffsw))

  # Compute total security
  sec = -math.log(epsilon, 2)

  # Return security, work, efficiency, and optimal probabilities
  return (sec, w, sec/w, xi)

# Challenge bits
n = 16
```

[4] http://cvxopt.org/

```
# Number of sampling points
N = 500

# Smallest possible value of epsilon
mineps = 2**(-n)

# Save data to a file by uniformly sampling values of epsilon
f = open('output%s.txt'%n, 'w')
plabel = '\t'.join(['p%s'%(i) for i in range(n+1)])
f.write('i\teps\ts\tw\tse\t%s\n'%plabel)

for i in range(N):
   s = float(i)/N * n
   e = 2**(-s)
   s, w, se, xi = solve_lp(e, n)
   xi = '\t'.join(map(str, xi))
   f.write('%s\t%s\t%s\t%s\t%s\t%s\n'%(i,e,s,w,se,xi))

f.close()
```