NO-LEAK AUTHENTICATION BY THE SHERLOCK HOLMES METHOD

DIMA GRIGORIEV AND VLADIMIR SHPILRAIN

"When you have eliminated the impossible, whatever remains, however improbable, must be the truth."
Arthur Conan Doyle, The Sign of Four

Abstract. We propose a class of authentication schemes that are *literally* zero-knowledge, as compared to what is formally defined as "zero-knowledge" in cryptographic literature. We call this "no-leak" authentication to distinguish from an established "zero-knowledge" concept. The "no-leak" condition implies "zero-knowledge" (even "perfect zero-knowledge"), but it is actually stronger, as we illustrate by examples.

The principal idea behind our schemes is: the verifier challenges the prover with questions that he (the verifier) already knows answers to; therefore, even a computationally unbounded verifier who follows the protocol cannot *possibly* learn anything new during any number of authentication sessions. This is therefore also true for a computationally unbounded passive adversary.

1. Introduction

For a general theory of public-key authentication (a.k.a. identification) as well as early examples of authentication protocols, the reader is referred to [10]. In this paper, we propose a class of challenge-response authentication schemes that are *literally* zero-knowledge by design, because the verifier knows the correct response *before* communicating his challenge to the prover, which means he cannot possibly obtain any new information from the prover's response. We call this "no-leak" authentication because the "zero-knowledge" terminology is "trademarked" in cryptographic literature by way of specific formal definitions, see e.g. [3]. Unfortunately, "zero-knowledge" (or even "perfect zero-knowledge") is just a euphemism, i.e., those formal definitions do not actually imply that there is no leak of information during any proof session. Even more unfortunately, it seems to be a rather common misconception these days that "perfect zero-knowledge" implies no leak for *any* honest verifier; this is what often happens with euphemisms. People who believe this, when confronted with a counterexample, usually respond with something like "No, no, there is no leak of information in a perfect zero-knowledge scheme". We leave these people alone and refer the readers interested in a more reasonable and scientific approach (than "No, no, there is no leak") to [11]

Research of the first author was partially supported by the Federal Agency of the Science and Innovations of Russia, State Contract No. 02.740.11.5192.

Research of the second author was partially supported by the NSF grants DMS-0914778 and DMS 1117675.

where an "hierarchy of zero-knowledge" is suggested, so that the presence of leak or lack thereof depends on the (honest) verifier's computational abilities.

Here we compare the established definition(s) of "perfect zero-knowledge" to our definition of "no-leak" that applies, incidentally, to a computationally unbounded forger as well. Perhaps the difference is best illustrated by scrutinizing the Graph ISO protocol from [4], which is known to be "perfect zero-knowledge", but it is obviously *not* "no-leak". We discuss this protocol in detail in our Section 4, while now we will try to explain the difference in definitions of "perfect zero-knowledge" vs. "no-leak". We note that there are several somewhat different definitions of "perfect zero-knowledge" (for example, the seminal paper [4] has 3 different definitions), so here we are just trying to capture most essential features of those definitions to help us make our point.

Definition 1. [4] **("perfect zero-knowledge")** *Let (P, V) be an interactive proof system for a language L. Let F be a polynomial-time probabilistic Turing machine (PTM) that produces forged transcripts. Let $\mathcal{T}(x)$ denote the set of all possible true transcripts (obtained by the verifier V actually engaging in an interactive proof with the prover P on input x), and $\mathcal{F}(x)$ the set of all possible forged transcripts. The proof system (P, V) is called perfect zero-knowledge (for L) if there exists a forger F such that for any $x \in L$:*

(i) *the set of forged transcripts is identical to the set of true transcripts, i.e., $\mathcal{F}(x) = \mathcal{T}(x)$;*

(ii) *the two associated probability distributions are identical, i.e., for any transcript $T \in \mathcal{T}(x)$, one has $Pr_{\mathcal{T}}[T] = Pr_{\mathcal{F}}[T]$.*

To better illustrate the difference between this and the following definition, we say, somewhat informally, that the former implies that in a perfect zero-knowledge proof system, if a specific leak of information (about the prover's secret key, say) can occur with probability p during an interaction between V and P, then it can occur with the same probability p if V does not interact with P at all, but just simulates such interaction.

By comparison, in the following definition the condition is (again, informally) that no leak should occur at all, i.e, the probability p alluded to in the previous paragraph is always 0. The following definition is an attempt to formalize this idea. Note also that in our definition, the "polynomial-time" restriction on the forger is dropped.

Definition 2. ("no-leak") *Let (P, V) be an interactive proof system for a language L. It is called no-leak if, for any $x \in L$ and for any function f from L to $\mathbf{N}$, any sequence $\mathcal{T}(x)$ of true transcripts obtained in time $\leq f(x)$ by interaction of the verifier V with the prover P on input x, could be also constructed, even by a deterministic algorithm, by V alone (i.e., without interacting with P) in time $\leq f(x)$.*

In other words, the prover can contribute nothing whatsoever to the verifier's ability of producing any sequence of true transcripts.

Probably the easiest way to achieve the "no-leak" property is to have the verifier V only ask those questions that he already knows the correct answers to. In this scenario,

the corresponding proof system will obviously be perfect zero-knowledge since whatever V can compute after interacting with the prover, he could already have computed *before* interacting with the prover. At the same time, "perfect zero-knowledge" does not necessarily imply "no leak"; as we have already mentioned, this is best illustrated by scrutinizing the graph ISO protocol from [4], so the reader who is not convinced at this point that the 'no-leak" condition is stronger than "perfect zero-knowledge", is referred to our Section 4.

It is well known that the main motivation for studying zero-knowledge proof systems is their application to authentication, which is also our main motivation here. The reason why we call our idea of authentication the "Sherlock Holmes method" is the following. It is the case with most natural decision problems in algebra (such as the identity problem, the conjugacy problem, the membership problem, etc.) that, for a generic input, the "no" answer can be obtained much more efficiently than the "yes" answer. Thus, if the prover "eliminates the impossible" by (efficiently) getting the "no" answers whenever she can, she is left with the only remaining possibility for a "yes" answer. We note that in a typical concrete realization of this idea, the prover will not be able to give a "yes" answer by any other method than "eliminating the impossible", which is why we call it the "Sherlock Holmes method."

To conclude the Introduction, we summarize what we think are the most interesting features of our proposal:

(1) A verifier who follows the protocol cannot possibly obtain from the prover any information that he does not already know. This is therefore also true for a passive adversary, even computationally unbounded one.
(2) There is no "concrete" problem for the adversary to solve in order to obtain the prover's long-term private key from her public key. The problem he/she faces is to obtain a test for non-membership in a set that he/she does not know.

Now a natural question is whether a verifier who does *not* follow the protocol (e. g. a cheating verifier) can obtain any information during an authentication session by offering challenges without knowing *a priori* what the correct response should be. We comment on this question in Section 3 where we answer several typical questions about our idea of "no-leak" proof in the form of a dialog with an inquisitive reader. We thought that arranging this material in the dialog format *after* describing our meta-protocol would be more helpful to the reader than if we made it part of the Introduction. Here we just say that our schemes are not any more insecure against cheating verifiers than existing authentication schemes are. It is just that the novelty of the present paper concerns the honest verifier scenario, whereas for the cheating verifier scenario we do not offer any new input.

Finally, we note that, in theory, our general scheme can also be used for encryption, see Remark 1 at the end of Section 2, although this is outside of the focus of our paper. In particular, we do not make any claims concerning security of relevant encryption schemes and do not discuss practical hardness assumptions. It should be understood that the focus of this paper is on theoretical aspects.

2. The meta-protocol

In this section, we give a description of our general idea of "authentication by the Sherlock Holmes method", leaving particular realizations to the next sections. Here Alice is the prover and Bob the verifier.

Alice's private key consists of: (a) a subset S_0 of some "universal" set S; (b) an efficient test telling that a given element of S does *not* belong to S_0; (c) a way to disguise S_0 to some S_0', e.g., a self-bijection φ of the universal set S, such that $\varphi(S_0) = S_0'$ and it is possible for Alice to efficiently compute both φ and φ^{-1}. In some realizations, (c) is not really necessary, and the role of S_0' is played just by a subset of S_0.

Alice's public key is a pair of sets S_0', S_1 that either are disjoint or have negligible intersection. *Note that the verifier does not know whether Alice has a "non-membership test" for S_0' or for S_1; this information is not public!* Both sets are given to the public in such a way that it is possible to efficiently select a random element from either set.

We note that the idea of a "non-membership test" for S_0 (see part (b) of Alice's private key above) can be expressed in a more precise language. Alice should have her private *separator* T, such that $S_0 \subset T$, the intersection $T \cap S_1$ is empty or negligible, and T is a "nice" set in the sense that the problem of membership in T is efficiently solvable. Thus, Alice can efficiently check membership of an element x in question in the set T; then, if $x \notin T$, she knows for sure that $x \notin S_0$. If $x \in T$, then she *assumes* that $x \in S_0$; the "tighter" (i.e., the closer to S_0) T is, the more chances this assumption has to be correct. Thus, even though there might be many different separators for a given pair of sets, a good separator is hard for the adversary to find without knowing the set S_0.

The protocol itself is the following sequence of steps.

(1) Bob selects a tuple $(x_1', \dots, x_{2m}')$ of random elements from either S_0' or S_1, with exactly m elements from S_0' and exactly m elements from S_1, and sends it to Alice.
(2) Alice checks, using her private test, for every i, whether for the element x_i corresponding to x_i', one has $x_i \notin S_0$. If her test fails, Alice assumes that $x_i \in S_0$ (equivalently, $x_i' \in S_0'$). Then she sends a tuple of $\frac{m}{2}$ "1"s to Bob, in $\frac{m}{2}$ randomly selected places corresponding to $x_i' \notin S_0'$. She gives no indication of the results of her test for the remaining $\frac{3m}{2}$ places.
(3) Bob, who knows the right answer, simply compares it to Alice's response and accepts or rejects authentication accordingly.

To prevent the adversary from guessing the right answer with non-negligible probability, several rounds of this protocol may be run (depending on what probability is accepted as "negligible"); this is similar to the Feige-Fiat-Shamir scheme [1]. We note that if, say, $m = 40$, then the probability of guessing the right answer in a single session is less than 10^{-6}.

Remark 1. *The protocol in this section can also be used for encryption. Namely, if Bob wants to transmit an encrypted bit to Alice, he sends her a random element from*

S'_0 in case he wants to transmit a "0", and a random element from S_1 in case he wants to transmit a "1".

As we have already pointed out in the Introduction, we do not pursue this direction here, and in particular, we do not make any claims concerning security of relevant encryption schemes.

2.1. **Correctness of the protocol.** The protocol is perfectly correct in the sense that if Alice is in possession of her private key and both parties follow all steps of the protocol, then Alice will be able to answer all Bob's challenges correctly with probability 1, and Bob will then accept Alice's authentication with probability 1. Even if the intersection $T \cap S_1$ is not empty and Alice's test fails for some elements sent by Bob, this will not affect Alice's response because her response only includes places where her test did not fail.

Also, since Bob does not obtain from Alice any information that he does not already know, he can construct by a deterministic algorithm any true authentication session transcript, without interacting with Alice. Thus, the following is obvious:

Proposition 1. *No information about the prover's private key is leaked during any authentication session with an honest verifier in the above authentication scheme, i.e., it is a "no-leak" proof system in the sense of Definition 2.*

This proposition should not create an impression that our authentication scheme is secure *only* in the honest verifier scenario. It is not any more insecure against cheating verifiers than existing authentication schemes are. It is just that the novelty of the present paper concerns the honest verifier scenario, whereas for the cheating verifier scenario we do not offer any new input. See our Section 3 for a somewhat more detailed discussion of this issue.

In Sections 5 and 6, we give two particular realizations of our meta-protocol just to illustrate the diversity of possible applications of our main idea. We emphasize once again though that we avoid detailed technical discussions (in particular, making quantitative statements) in this paper and keep the focus on theoretical aspects.

3. Questions and answers

In this section, we answer some questions/concerns that a reader may have at this point; these questions have actually been asked by some of our colleagues. We thought that arranging this material in the dialog format *after* describing our meta-protocol would be more helpful to the reader than if we made it part of the Introduction.

Q. Do you prove that recovering the prover's private key from her public key in any of your particular realizations is computationally infeasible? In particular, do you prove the existence of one-way functions?

A. No and no. Moreover, our focus in this paper is on leak (or lack thereof) of information during authentication session, rather than on security of the prover's public key construction.

Q. If not, then what is the novelty of your proposal?

A. The main novelty is that in our authentication scheme, a verifier who follows the protocol cannot possibly obtain from the prover any information that he does not already know. This is also true for a passive adversary, even computationally unbounded one. (Of course, a computationally unbounded adversary can usually find correct responses to the verifier's challenges by using "brute force", but this is a different story.)

Q. In that case, what is the difference between your protocol and the classical zero-knowledge Graph Non-ISO protocol [5] where the prover convinces the verifier that two given graphs are non-isomorphic by giving to the verifier only the information that he already knows?

A. In the Graph Non-ISO protocol [5] the prover is assumed to be computationally unbounded (i.e., there is no "trapdoor"). Therefore, the Graph Non-ISO protocol cannot be used for authentication purposes in any meaningful scenario in real life.

Q. Well then, how about well-known zero-knowledge protocols with trapdoor, such as Graph ISO, or quadratic residuosity, or Feige-Fiat-Shamir scheme?

A. None of these protocols is "no-leak"; we think it is best illustrated by analyzing the Graph ISO protocol; this is what we do in the next Section 4.

Q. What about a verifier who does *not* follow the protocol? What if he just produces some random challenges? Can he obtain some information about the prover's private key from her responses in that case?

A. The answer to this question may depend on a particular realization of our meta-protocol; more specifically, on the size of the sets S'_0 and S_1 relative to the size of the "universal" set S from which a "frivolous" verifier can pick up his random challenges. Typically, the sets S'_0 and S_1 are negligible in S, which implies that if a frivolous verifier picks his challenges randomly from S, he will get the "not in S_0" response for every random element from S. Whether or not this can give him any useful information about S_0 other than S_0 is negligible in S, again depends on a particular protocol, but this question has been well studied in the context of zero-knowledge schemes in the "usual" sense, and we do not offer any new insight into this question in the present paper.

Q. Does this mean that your authentication schemes are insecure against cheating verifiers?

A. No, our schemes are not any more insecure against cheating verifiers than existing authentication schemes are. It is just that the novelty of the present paper concerns the honest verifier scenario, whereas for the cheating verifier scenario we do not offer any new input.

Q. Well then, how useful is your new input in real life? Isn't it true that in real life, the cheating verifier scenario is prevalent?

A. Not really. The honest verifier scenario is used by millions of people every day while doing Internet shopping or banking (provided, of course, they use trusted bankers or

merchants, which is what most people do). It is true that occasionally, during Internet shopping, there might be a concern about the verifier being honest. What usually happens is that the remote host has to authenticate itself to your computer first, in which case your computer plays the role of an honest verifier. Thus, in this case the protocol is just a combination of two authentication protocols with honest verifiers. Of course, in non-commercial (e.g. military) applications, a situation where the verifier is malicious is not so unusual, but it appears that in real life in general, authentication with an honest verifier is still prevalent.

4. Why is the Graph ISO proof system not "no-leak"?

In this section, we try to resolve the confusion that stems from taking the "perfect zero-knowledge" euphemism too literally. To that end, we recall the well-known Graph ISO protocol from [4]. Here Alice is the prover and Bob the verifier.

(1) Alice's public key consists of two isomorphic graphs, Γ_0 and Γ_1. Alice's private key is an isomorphism $\varphi : \Gamma_1 \to \Gamma_0$. Alice is supposed to prove to Bob that Γ_0 is isomorphic to Γ_1.
(2) To begin a session, Alice selects a random bit b, a random isomorphism $\sigma : \Gamma_b \to H$, and sends the "commitment" graph H to Bob.
(3) Bob chooses a random bit c and sends it to Alice.
(4) Alice responds with an isomorphism $\tau : \Gamma_c \to H$.
(5) Bob verifies that τ is, indeed, an isomorphism.

Obviously, if the same graph H appears as commitment in two different sessions, and if the corresponding challenge bits c are different, then Bob, who gets isomorphisms $\tau_0 : \Gamma_0 \to H$ and $\tau_1 : \Gamma_1 \to H$, can recover the isomorphism $\tau_0^{-1}\tau_1$ between Γ_1 and Γ_0. Therefore, after two sessions, Bob can obtain, with non-zero probability, information that he did not have before he started to interact with Alice. Thus, it is easy to see that the above proof system is not "no-leak" in the sense of Definition 2. Indeed, if interaction between Bob Alice produces a sequence of, say, just two transcripts with the same commitment graph H and different corresponding challenge bits c, then to reproduce this sequence with probability 1 by a deterministic algorithm, Bob would need a lot more than just 2 attempts.

At the same time, this proof system is "perfect zero-knowledge" in the sense of Definition 1 because Bob can obtain the same information *with the same probability* (but not with probability 1!) if he simulates the protocol by himself, without interacting with Alice. Specifically, by selecting random isomorphisms $\alpha : \Gamma_0 \to H_1$ and $\beta : \Gamma_1 \to H_2$, he may end up, with non-zero probability, with $H_1 = H_2$, hence recovering an isomorphism between Γ_1 and Γ_0.

5. A particular realization: subset sum

In this section, we offer a particular realization of the meta-protocol from Section 2, exploiting the hardness of the subset sum problem, see e.g. [2]. We note that the complexity of this particular problem was previously used in [7] in different cryptographic

contexts, namely for constructing a pseudo-random generator and a universal one-way hash function.

The "universal" set S in this section is the set of all m-tuples of m-dimensional vectors over $\mathbf{Q}$.

Alice's private key is a set $S_0 = \{a_1, \ldots, a_m\}$ of m random linearly independent (over $\mathbf{Q}$) m-dimensional vectors *with integer coordinates*, which is therefore a basis of $\mathbf{Q}^m$. However, the vector a_1 is special: the g.c.d. of its coordinates is even.

Alice's public key includes the vector a_1 and a set of $k > m$ random vectors $c_1, \ldots, c_k$ from the $\mathbf{Z}_+$-span of S_0.

Now we give a description of the authentication protocol. *To simplify the notation, we give an exposition where Bob challenges Alice with just a single element rather than with a tuple of elements, as in the meta-protocol in Section 2.*

(1) Bob selects, with equal probabilities (see our subsection 5.1 for details), either a random vector $c \in Span_{\mathbf{Z}_+}(a_1, c_1, \ldots, c_k)$ or a random vector $c \in Span_{\mathbf{Z}_+}(\frac{1}{2}a_1, c_1, \ldots, c_k)$ and sends the vector c to Alice; in the latter case, Bob takes care that the coefficient at $\frac{1}{2}a_1$ is odd. Here $Span_{\mathbf{Z}_+}$ denotes the set of all linear combinations of given vectors *with nonnegative integer coefficients*.

(2) Alice, using standard linear algebra, finds (rational) coordinates of c in the basis S_0. If at least one of these coordinates is not a nonnegative integer, she knows that $c \notin Span_{\mathbf{Z}_+}(a_1, c_1, \ldots, c_k)$; therefore, she sends "1" to Bob. If all coordinates are nonnegative integers, Alice assumes that $c \in Span_{\mathbf{Z}_+}(a_1, c_1, \ldots, c_k)$, and sends "0" to Bob.

(3) Bob, who knows the right answer, simply compares it to Alice's response and accepts or rejects authentication accordingly.

We note that there is a negligible probability for Bob to reject a legitimate Alice because it may happen that all coordinates of c in the basis S_0 are nonnegative integers, but $c \notin Span_{\mathbf{Z}_+}(a_1, c_1, \ldots, c_k)$. It may, in fact, even happen (again, with negligible probability) that $c \in Span_{\mathbf{Z}_+}(a_1, c_1, \ldots, c_k)$, but Bob expected Alice to respond with a "1" because he selected his $c \in Span_{\mathbf{Z}_+}(\frac{1}{2}a_1, c_1, \ldots, c_k)$.

We also note that the reason for using a public vector a_1 with g.c.d. of coordinates even is to have Bob's vector c in $Span_{\mathbf{Q}_+}(a_1, c_1, \ldots, c_k)$ in either case, because there is a polynomial-time test detecting whether or not a given vector belongs to the $\mathbf{Q}_+$-span of other given vectors (cf. linear programming problem), see [8] or [12].

Finally, we note that the problem faced by the adversary who wants to impersonate the prover is the following: find out whether or not the matrix equation $Bx = c$ has a solution for x as a vector with nonnegative integer coordinates. Here B is the matrix made up of coordinates of the vectors $a_1, c_1, \ldots, c_k$, c is the challenge vector selected by Bob, and x is the vector unknown to both the prover and the adversary. A special case of this problem, where B is just a vector with integer coordinates, x is a 0-1 vector, and c is just an integer, is known as the *subset sum problem* and is NP-complete, see e.g. [2]. Moreover, as pointed out, for example, in [3, p.41], it appears that the subset sum problem might be hard on random instances, not just on some carefully selected ones.

5.1. **Suggested parameters and key generation.** Suggested parameter values for the protocol above are:

(1) The dimension of vectors is $m = 20$.
(2) Coordinates of the vectors a_i: random nonnegative integers ≤ 10. We note that m random m-dimensional vectors like that are going to be linearly independent with overwhelming probability.
(3) Vectors c_i are constructed by Alice as random linear combinations of the vectors a_i with nonnegative integer coefficients ≤ 10. The number of vectors c_i is $k = 2m$.
(4) At step (1) of the protocol, Bob constructs his vector c as a random linear combination of the public vectors $a_1, c_1, \ldots, c_k$ with nonnegative integer coefficients ≤ 10, with one possible exception: according to the protocol description, he may choose the coefficient at a_1 to be of the form $\frac{n}{2}$, where n is odd, $1 \leq n \leq 19$.

6. A particular realization: polynomial equations

In this section, we offer another particular realization of the meta-protocol from Section 2.

Alice's private key consists of: (i) a large prime $p \equiv 3 \ (mod\ 4)$; (ii) two random polynomials $h(x_1, \ldots, x_k)$ and $g(x_1, \ldots, x_k)$ over $\mathbf{Z}_p$; (iii) a random constant $c \in \mathbf{Z}_p$.

Alice's public key includes: (i) polynomial $f(x_1, \ldots, x_k) = (h(x_1, \ldots, x_k))^2 - c \ (mod\ p)$. (Polynomial f is published as a polynomial over $\mathbf{Z}$, without specifying p.) Thus, for any $x_1, \ldots, x_k \in \mathbf{Z}$, there is $u \in \mathbf{Z}$ such that $f(x_1, \ldots, x_k) + c = u^2 (mod\ p)$; (ii) polynomial $s(x_1, \ldots, x_k) = -((g(x_1, \ldots, x_k))^2 + 1)^2 - c \ (mod\ p)$. (Again, polynomial s is published as a polynomial over $\mathbf{Z}$, without specifying p.) Thus, a value of the polynomial $s(x_1, \ldots, x_k) + c$ is never a square modulo p because -1 is not a square modulo p (since $p \equiv 3 \ (mod\ 4)$), and $(g(x_1, \ldots, x_k))^2 + 1$ is never equal to 0 modulo p, for the same reason.

Now we give a description of the authentication protocol. Again, *to simplify the notation, we give an exposition where Bob challenges Alice with just a single element rather than with a tuple of elements, as in the meta-protocol in Section 2.*

(1) Bob selects (see our subsection 6.1 for details) random integers $x_1, \ldots, x_k$ and plugs them, with equal probabilities, into either f or s. He then sends the result, call it $Bob(x_1, \ldots, x_k)$, to Alice.
(2) Alice computes $a = Bob(x_1, \ldots, x_k) + c \ (mod\ p)$ and checks whether or not a is a square modulo p. If not, she knows that $Bob(x_1, \ldots, x_k) \neq f(x_1, \ldots, x_k)$ and sends "1" to Bob. If it is, Alice assumes that $Bob(x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$ and sends "0" to Bob.
(3) Bob, who knows the right answer, simply compares it to Alice's response and accepts or rejects authentication accordingly.

The way Alice checks whether or not a is a square modulo p is as follows. She raises a to the power of $\frac{p-1}{2}$. If the result is equal to 1 modulo p, then a is a square modulo p; if not, then it is not.

6.1. **Suggested parameters and key generation.** Suggested parameter values for the protocol above are:

(1) The number k of variables: between 3 and 5.
(2) The value of p: on the order of 2^t, where t is the security parameter.
(3) The degree of Alice's private polynomials h, g: between 2 and 3.
(4) Bob generates his integers $x_1, \ldots, x_k$ uniformly randomly from the interval $[1, 2^{\frac{t}{k}}]$.

Remark 2. *The adversary may try to attack Bob's challenge by solving one of the equations* $f(x_1, \ldots, x_k) = Bob(x_1, \ldots, x_k)$ *or* $s(x_1, \ldots, x_k) = Bob(x_1, \ldots, x_k)$ *for integers* $x_1, \ldots, x_k$*, or just try to find out whether either of these equations has integer solutions. The corresponding decision problem (the Diophantine problem, or Hilbert's 10th problem) is known to be undecidable, see* [9]*. In our situation, however, adversary actually faces a promise problem since he/she knows that at least one of the equations has integer solutions. Furthermore, in our situation the range for the unknowns is bounded. Still, the "bounded" Diophantine problem is known to be NP-hard, see e.g.* [2]*, which makes this kind of attack look infeasible, although we avoid making such claims in this paper, as was explained in the Introduction.*

Acknowledgement. Both authors are grateful to Max Planck Institut für Mathematik, Bonn for its hospitality during the work on this paper. We are also grateful to Nelly Fazio and William E. Skeith for useful discussions.

References

[1] U. Feige, A. Fiat and A. Shamir, *Zero knowledge proofs of identity,* Journal of Cryptology **1** (1987), 77–94.
[2] M. Garey, J. Johnson, *Computers and Intractability, A Guide to NP-Completeness*, W. H. Freeman, 1979.
[3] O. Goldreich, *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2007.
[4] O. Goldreich, S. Micali, A. Wigderson, *Proofs that Yield Nothing but their Validity, or All Languages in NP have Zero-Knowledge Proof Systems*, J. ACM **38** (1991), 691–729.
[5] S. Goldwasser, S. Micali, and C. Rackoff, *The Knowledge Complexity of Interactive Proof Systems*, SIAM J. Comput. **18** (1) (1989), 186-208.
[6] D. Grigoriev, I. Ponomarenko, *Constructions in public-key cryptography over matrix groups*, Contemp. Math., Amer. Math. Soc. **418** (2006), 103–119.
[7] R. Impagliazzo, M. Naor, *Efficient cryptographic schemes provably as secure as subset sum*, J. Cryptology **9** (1996), 199–216.
[8] L. G. Khatchyian, *A polynomial algorithm in linear programming*, Doklady Akad. Nauk USSR, **244** (1979), 1093–1096 (Russian). [Translated as Soviet Math. Doklady **20** (1979), 191–194.]
[9] Yu. Matiyasevich, *Hilbert's 10th Problem (Foundations of Computing)*, The MIT Press, 1993.
[10] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC-Press 1996.
[11] O. Pandey, R. Pass, A. Sahai, W. Tseng and M. Venkitasubramaniam, *Precise concurrent zero knowledge*, in: Eurocrypt 2008, Lecture Notes Comp. Sc. **4965** (2008), 397–414.
[12] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley 1998.

CNRS, Mathématiques, Université de Lille, 59655, Villeneuve d'Ascq, France
E-mail address: dmitry.grigoryev@math.univ-lille1.fr

Department of Mathematics, The City College of New York, New York, NY 10031
E-mail address: shpil@groups.sci.ccny.cuny.edu