Authentication schemes from actions on graphs, groups, or rings

Dima Grigoriev

CNRS, Mathématiques, Université de Lille, 59655, Villeneuve d'Ascq, France

Vladimir Shpilrain

Department of Mathematics, The City College of New York, New York, NY 10031

To Nikolai Alexandrovitch Shanin with deepest appreciation

Abstract

We propose a couple of general ways of constructing authentication schemes from actions of a semigroup on a set, without exploiting any specific algebraic properties of the set acted upon. Then we give several concrete realizations of this general idea, and in particular, we describe several authentication schemes with long-term private keys where forgery (a.k.a. impersonation) is NP-hard. Computationally hard problems that can be employed in these realizations include Graph Colorability, Diophantine Problem, and many others.

Keywords: authentication, one-way function, NP-hard

1. Introduction

In this paper, we propose several general Feige-Fiat-Shamir-like [3] constructions of authentication schemes (with long-term private keys) from arbitrary actions. For a general theory of public-key authentication (a.k.a. iden-

*corresponding author

Email addresses: `dmitry.grigoryev@math.univ-lille1.fr` (Dima Grigoriev), `shpil@groups.sci.ccny.cuny.edu` (Vladimir Shpilrain*)

[1]Research of the second author was partially supported by the NSF grant DMS-0405105.

tification) as well as early examples of authentication protocols, the reader is referred to [10].

Suppose a (partial) semigroup S acts on a set X, i.e., for $s, t \in S$ and $x \in X$, one has $(st)(x) = s(t(x))$ whenever both sides are defined. For cryptographic purposes, it is good to have an action which is "hard-to-invert". We deliberately avoid using the "one-way function" terminology here because we do not want to be distracted by formal definitions that are outside of the main focus of this paper. For a rigorous definition of a one-way function, we just refer to one of the well-established sources, such as [5]. It is sufficient for our purposes to use an intuitive idea of a hard-to-invert action which is as follows. Let X and Y be two sets such that complexity (or "size") $|u|$ is defined for all elements u of either set. A function $f : X \to Y$ is hard-to-invert if computing $f(x)$ takes time polynomial in $|x|$ for any $x \in X$ (which implies, in particular, that complexity of $f(x)$ is bounded by a polynomial function of $|x|$), but there is no known algorithm that would compute some $f^{-1}(y)$ in polynomial time in $|y|$ for every $y \in f(X)$.

In our context of actions, we typically consider hard-to-invert functions of the type $f_x : s \to s(x)$; in particular, a secret is usually a *mapping* s, which makes our approach different from what was considered before. This idea allows us to construct several general Feige-Fiat-Shamir-like authentication schemes (with long-term private keys) from arbitrary actions, see Section 3. Then, in the subsequent sections, we give several concrete realizations of this general idea, and in particular, we describe several authentication schemes where recovering the prover's long-term private key from her public key is an NP-hard problem. We note however that what really matters for cryptographic security is computational intractability of a problem on a *generic* set of inputs, i.e., the problem should be hard on "most" randomly selected inputs. For a precise definition of the "generic-NP" class, we refer to [11]. Here we just say that some of the problems that we employ in the present paper, e.g. Graph Colorability, are *likely* to be generically NP-hard, which makes them quite attractive for cryptographic applications.

We also address an apparently easier task of *forgery* (a.k.a. *misrepresentation*, a.k.a. *impersonation*), and show that in most of our schemes this, too, is equivalent for the adversary to solving an NP-hard problem. To be more specific, by *forgery* we mean the scenario where the adversary enters the authentication process at the *commitment* step, and then has to respond to the *challenge* properly.

Finally, we note that there were other attempts at constructing authentication schemes based on NP-hard problems (e.g. [1], [2]), but these constructions are less transparent, and it is not immediately clear how or why they work.

2. When a composition of functions is hard-to-invert

Here we prove a simple but useful proposition about composing a hard-to-invert function with another function, which is not necessarily hard-to-invert.

Proposition 1. *Let A, B and C be sets, and let $\varphi : A \to B$ and $\psi : B \to C$ be two functions such that computing $\varphi(x)$ as well as $\psi(x)$ takes time polynomial in $|x|$ for any x for which the corresponding function is defined.*

(a) *If ψ is hard-to-invert and the domain of ψ is contained in the range of φ, then the composition $\varphi\psi = \psi(\varphi)$ is hard-to-invert.*

(b) *If φ is hard-to-invert, ψ is injective (i.e., one-to-one), and the domain of ψ contains the range of φ, then the composition $\varphi\psi = \psi(\varphi)$ is hard-to-invert.*

Proof. **(a)** Let $f = \psi(\varphi) : A \to C$. By way of contradiction, suppose there is an algorithm $\mathcal{A}$ that computes some $f^{-1}(c)$ in polynomial time in $|c|$ for every $c \in f(A)$.

Now let $y \in \psi(B)$. Since the domain of ψ is contained in the range of φ, this implies $y \in f(A)$. Then we apply the algorithm $\mathcal{A}$ to y to get some $a \in A$. This takes polynomial time in $|y|$, and, in particular, the size of a is polynomial in $|y|$. Then we apply φ to a to get an element $b \in B$. This takes polynomial time in $|a|$, and therefore also in $|y|$. Since now $\psi(b) = y$, we have found a preimage of y under ψ in polynomial time in $|y|$, contradicting the assumption on ψ to be hard-to-invert.

(b) Again, let $f = \psi(\varphi) : A \to C$ and suppose, by way of contradiction, that there is an algorithm $\mathcal{A}_1$ that computes some $f^{-1}(c)$ in polynomial time in $|c|$ for every $c \in f(A)$.

Now let $y \in \varphi(A)$. Since the domain of ψ contains the range of φ, we can apply ψ to y to get $\psi(y) = c \in C$. This takes polynomial time in $|y|$. Then we apply the algorithm $\mathcal{A}_1$ to c to obtain some $a = f^{-1}(c)$. This takes polynomial time in $|c|$, and therefore also in $|y|$. Now we claim that $\varphi(a) = y$, for if it was not the case, we would have $y_1 \neq y$ such that $\psi(y) = \psi(y_1) = c$ (since $f(a)$ should be equal to c), contradicting the assumption on ψ to be injective.

$\square$

3. Three protocols

In this section, we give a description of three generic authentication protocols (or, rather, "meta-protocols", or "primitives", since we do not give any implementation details in this section). Here Alice is the prover and Bob the verifier.

3.1. Protocol I

Suppose a set S acts on a set X, i.e., for any $s \in S$ and $x \in X$, the element $s(x) \in X$ is well-defined.

1. Alice's public key consists of a set X, a (partial) semigroup S, an element $x \in X$, and an element $u = s(x)$ for some randomly selected $s \in S$; this u is her long-term private key.
2. To begin authentication, Alice selects an element $t \in S$ and sends the element $v = t(s(x)) \in X$, called the *commitment*, to Bob.
3. Bob chooses a random bit c, called the *challenge*, and sends it to Alice.
   - If $c = 0$, then Alice sends the element t to Bob, and Bob checks if the equality $v = t(u)$ is satisfied. If it is, then Bob accepts the authentication.
   - If $c = 1$, then Alice sends the composition ts to Bob, and Bob checks if the equality $v = ts(x)$ is satisfied. If it is, then Bob accepts the authentication.

3.2. Protocol II

Yet another protocol involving a composition of actions (or mappings) is as follows.

1. Alice's public key consists of a set X, a (partial) semigroup S whose elements may act on X, an element $x \in X$, and an element $z = r(x)$ for some randomly selected $r \in S$; this z is her long-term private key.
2. To begin authentication, Alice selects an element $y \in X$, together with two elements $s, t \in S$ such that $s(x) = y$ and $t(y) = z$. She then sends the element y (the commitment) to Bob.
3. Bob chooses a random bit c, the challenge, and sends it to Alice.
   - If $c = 0$, then Alice sends the element s to Bob, and Bob checks if the equality $s(x) = y$ is satisfied. If it is, then Bob accepts the authentication.

- If $c = 1$, then Alice sends the element t to Bob, and Bob checks if the equality $t(y) = z$ is satisfied. If it is, then Bob accepts the authentication.

We note that selecting an element y at the commitment step of this protocol may be non-trivial; later in this paper we show how to implement this step in particular realizations of Protocol II.

Proposition 2. *Suppose that after several runs of steps (2)-(3) of the above Protocol II, both values of c are encountered. Then successful forgery in such a protocol is equivalent to compromising Alice's long-term private key, i.e., to finding $r' \in S$ such that $z = r'(x)$.*

Proof. Suppose Eve wants to impersonate Alice. To that effect, she interferes with the commitment step by sending her own commitment $y' \in X$ to Bob, such that $s'(x) = y'$ and $t'(y') = z$ for some $s', t' \in S$. Since she should be prepared to respond to both challenges $c = 0$ and $c = 1$, she should be able to produce s' as well as t'. Therefore, she is able to also produce their composition $t's'$. The result now follows from: $z = t'(y') = t'(s'(x)) = (t's')(x)$, so that $r' = t's'$. $\square$

3.3. Protocol III

In this protocol, the hardness of obtaining the long-term private key for the adversary can be based on "most any" search problem; we give some concrete examples in the following sections, whereas in this section, we give a generic protocol.

1. Alice's public key consists of a set S that has a property $\mathcal{P}$. Her long-term private key is a *proof* (or a "witness") that S does have this property. We are also assuming that the property $\mathcal{P}$ is preserved by *isomorphisms.*
2. To begin authentication, Alice selects an isomorphism φ that can be applied to S, and sends the set $S_1 = \varphi(S)$ (the commitment) to Bob.
3. Bob chooses a random bit c and sends it to Alice.
   - If $c = 0$, then Alice sends the isomorphism φ to Bob, and Bob checks (i) if $\varphi(S) = S_1$ and (ii) if φ is an isomorphism.
   - If $c = 1$, then Alice sends a proof of the fact that S_1 has the property $\mathcal{P}$ to Bob, and Bob checks its validity.

The following proposition says that in the Protocol III, successful forgery is equivalent for the adversary to finding Alice's private key from her public key, which is equivalent, in turn, to giving a proof (or a "witness") that S does have the property $\mathcal{P}$. The latter problem can be selected from a large pool of NP-hard problems (see e.g. [4]).

Proposition 3. *Suppose that after several runs of steps (2)-(3) of the above Protocol III, both values of* c *are encountered. Then successful forgery in such a protocol is equivalent to finding a proof of the fact that* S *has the property* $\mathcal{P}$.

Proof. Suppose Eve wants to impersonate Alice. To that effect, she interferes with the commitment step by sending her own commitment S_1' to Bob. Since she should be prepared to respond to the challenge $c = 0$, she should know an isomorphism $\varphi' : S \to S_1'$. On the other hand, since she should be prepared for the challenge $c = 1$, she should know a proof of the fact that S_1' has the property $\mathcal{P}$. Therefore, since φ' is invertible, this implies that she can produce a proof of the fact that S has the property $\mathcal{P}$. This completes the proof in one direction.

The other direction is trivial. $\square$

Remark 1. *We note that finding a proof of the fact that a given* S *has a property* $\mathcal{P}$ *is not a decision problem, but rather a search problem (sometimes also called a* promise *problem), so we cannot formally allocate it to one of the established complexity classes. However, we observe that, if there were an algorithm* $\mathcal{A}$ *that would produce, for any* S *having a property* $\mathcal{P}$*, a proof of that fact in time bounded by a polynomial* $P(|S|)$ *in the "size"* $|S|$ *of* S*, then, given an arbitrary* S'*, we could run the algorithm* $\mathcal{A}$ *on* S'*, and if it would not produce a proof of* S' *having the property* $\mathcal{P}$ *after running over the time* $P(|S'|)$*, we could conclude that* S' *does not have the property* $\mathcal{P}$*, thereby solving the corresponding decision problem in polynomial time.*

4. Subgraph isomorphism (Protocol II)

There is a classical realization of the Protocol I from Section 3 (actually, it also fits in with the Protocol III), based on the Graph Isomorphism problem, see [6]. We note that this *decision* problem is in the class NP, but it is not known to be NP-hard. Moreover, generic instances of this problem are easy, because two random graphs are typically non-isomorphic for trivial reasons. However, the problem that is actually used in [6] is a *promise* problem: given two isomorphic graphs, find a particular isomorphism between them. This is not a decision problem; therefore, if we are to allocate it to one of the established complexity classes, we need some kind of "stratification" to convert it to a decision problem. This can be done as follows. Any isomorphism of a graph Γ on n vertices can be identified with a permutation of the tuple $(1, 2, \ldots, n)$, i.e., with an element of the symmetric group S_n. If we choose a set of generators $\{g_i\}$ of S_n, we can ask whether or not there is an isomorphism between two given graphs Γ and Γ_1, which can be represented as a product of at most k generators g_i. To the best of our knowledge, the question of NP-hardness of this problem has not been addressed in the literature, but it looks like a really interesting and important problem.

Anyway, in this section, we describe a realization of the Protocol II from Section 3, based on the Subgraph Isomorphism problem. It is very similar to the Graph Isomorphism problem, but unlike the Graph Isomorphism problem, it is *known* to be NP-hard, see e.g. [4, Problem GT48]. We also note that this problem contains many other problems about graphs, including the Hamiltonian Circuit problem, as special cases. The Subgraph Isomorphism problem is: given two graphs Γ_1 and Γ_2, find out whether or not Γ_1 is isomorphic to a subgraph of Γ_2.

1. Alice's public key consists of two graphs, Γ and Γ_2. Alice's private key is a subgraph Γ_1 of Γ_2 and an isomorphism $\varphi : \Gamma \to \Gamma_1$.
2. To begin authentication, Alice selects an "intermediate" graph Λ, which is a subgraph of Γ_2, and an isomorphic embedding $\psi : \Gamma \to \Lambda$, with $\psi(\Gamma) = \Gamma_1$. Then she sends the graph Λ (the commitment) to Bob, while keeping the embeddings $\psi : \Gamma \to \Lambda$ and $\tau : \Lambda \to \Gamma_2$ to herself.
3. Bob chooses a random bit c and sends it to Alice.
   - If $c = 0$, then Alice sends the embedding ψ to Bob, and Bob checks if ψ is actually an embedding of Γ into Λ.
   - If $c = 1$, then Alice sends the embedding τ to Bob, and Bob checks if τ is actually an embedding of Λ into Γ_2.

We point out here the following corollary to our Proposition 2:

Corollary 1. *Suppose that after several runs of steps (2)-(3) of the above protocol, both values of c are encountered. Then successful forgery in such a* *protocol is equivalent to compromising Alice's long-term private key, i.e., to finding an embedding φ' of Γ into Γ_2.*

We note that the problem alluded to at the end of this corollary (the Subgraph Isomorphism problem) is NP-complete, see e.g. [4, Problem GT48].

A few more comments are in order.

- As it is usual with Feige-Fiat-Shamir-like authentication protocols, steps (2)-(3) of this protocol have to be iterated several times to prevent a successful forgery with non-negligible probability.

- When we say that Alice "sends" (or "publishes") a graph, that means that Alice sends or publishes its adjacency matrix. Thus, the size of Alice's public key is roughly $2n^2$, where n is the number of vertices in Γ.

- When we say that Alice "sends a subgraph" of a bigger graph, that means that Alice sends the numbers $\{m_1, m_2, \ldots, m_n\}$ of vertices that define this subgraph in the bigger graph. When she sends such a subgraph together with an isomorphism from another (sub)graph, she sends a map $(k_1, k_2, \ldots, k_n) \to (m_1, m_2, \ldots, m_n)$ between the vertices.

- Alice can construct the "intermediate" graph Λ at Step 2 of the protocol by simply discarding some randomly selected vertices (together with incident edges) of the graph Γ_2 that do not belong to Γ_1. Since Alice knows an embedding of Γ into Γ_2, she will then know an embedding of Γ into Λ, too.

5. Graph homomorphism (Protocol I)

In this section, we use the Graph Homomorphism problem that is known to be NP-complete, see [4, Problem GT52]. We have to briefly describe this problem first.

Given two graphs, Γ_1 and Γ_2, the Graph Homomorphism problem asks whether or not there is a homomorphism $f : \Gamma_1 \to \Gamma_2$, i.e., a mapping from the vertex set of Γ_1 onto the vertex set of Γ_2 such that for any two adjacent vertices v_1, v_2 of Γ_1, their images $f(v_1)$ and $f(v_2)$ are adjacent in Γ_2. We note that the Graph Homomorphism problem remains NP-complete even if Γ_2 is a triangle, see [4, Problem GT52].

Now the authentication protocol is as follows.

1. Alice's public key consists of two graphs, Γ_1 and Γ_2. Alice's long-term private key is a homomorphism $\alpha : \Gamma_1 \to \Gamma_2$.
2. To begin authentication, Alice selects a graph Γ together with a homomorphism $\beta : \Gamma \to \Gamma_1$ and sends the graph Γ (the commitment) to Bob, while keeping β to herself.
3. Bob chooses a random bit c and sends it to Alice.
   - If $c = 0$, then Alice sends the homomorphism β to Bob, and Bob checks whether $\beta(\Gamma) = \Gamma_1$ and whether β is a homomorphism (i.e., whether β takes adjacent vertices to adjacent ones).
   - If $c = 1$, then Alice sends the composition $\alpha\beta = \beta(\alpha)$ to Bob, and Bob checks whether $\alpha\beta(\Gamma) = \Gamma_2$ and whether $\alpha\beta$ is a homomorphism.

We now give a couple of comments on the above protocol.

- To generate her public key, Alice starts with a random graph Γ_2 and constructs Γ_1 as follows. She selects randomly a subset V' of the vertex set of Γ_2, and for each vertex v from V' does the following. First, she replaces v by several new vertices $u_1, \ldots, u_k$. These vertices are going to be mapped onto the vertex v by the homomorphism that Alice is trying to construct. Thus, Alice arbitrarily connects each u_i to some other vertices adjacent to v. She repeats this procedure with each vertex from V' and obtains her private homomorphism α as a composition of intermediate homomorphisms.

  The same way Alice can construct a graph Γ from Γ_1 at the commitment step.

- We note that, instead of trying to find a homomorphism between given graphs, Eve can try to find any graph Γ' that would map homomorphically onto both Γ_1 and Γ_2, together with the corresponding homomorphisms. Then she can interfere at the commitment step and send this Γ' to Bob, which will allow her to respond to either challenge by Bob successfully. The problem of finding such a graph Γ' (a "common multiple" of two given graphs, so to speak) is of independent interest. We do not know whether it has been previously addressed in the literature.

6. Graph colorability (Protocol III)

Graph colorability (more precisely, k-colorability) appears as problem [GT4] on the list of NP-complete problems in [4]. We include an authentication protocol based on this problem here as a special case of the Protocol III from Section 3. We note that a (rather peculiar) variant of this problem was shown to be NP-hard *on average* in [16] (the latter paper deals with edge coloring though). As we have pointed out in our Section 3, "most any" search problem can be used in Protocol III; we choose the graph colorability problem here just to illustrate this point, i.e., we do not claim that this is the best choice of underlying problem in terms of security, say.

1. Alice's public key is a k-colorable graph Γ, and her private key is a k-coloring of Γ, for some (public) k.
2. To begin authentication, Alice selects an isomorphism $\psi : \Gamma \to \Gamma_1$, and sends the graph Γ_1 (the commitment) to Bob.
3. Bob chooses a random bit c and sends it to Alice.
    - If $c = 0$, then Alice sends the isomorphism ψ to Bob. Bob verifies that ψ is, indeed, an isomorphism from Γ onto Γ_1.
    - If $c = 1$, then Alice sends a k-coloring of Γ_1 to Bob. Bob verifies that this is, indeed, a k-coloring of Γ_1.

Again, a couple of comments are in order.

- It is obvious that if Γ is k-colorable and Γ_1 is isomorphic to Γ, then Γ_1 is k-colorable, too.
- When we say that Alice "sends a k-coloring", that means that Alice sends a set of pairs (v_i, n_i), where v_i is a vertex and n_i are integers between 1 and k such that, if v_i is adjacent to v_j, then $n_i \neq n_j$.
- Alice's algorithm for creating her public key (i.e., a k-colorable graph Γ) is as follows. First she selects a number n of vertices; then she partitions n into a sum of k positive integers: $n = n_1 + \ldots + n_k$. Now the vertex set V of the graph Γ will be the union of the sets V_i of cardinality n_i. No two vertices that belong to the same V_i will be adjacent, and any two vertices that belong to different V_i will be adjacent with probability $\frac{1}{2}$. The k-coloring of Γ) is then obvious: all vertices in the set V_i are colored in color i.

Proposition 4. *Suppose that after several runs of steps (2)-(3) of the above protocol, both values of c are encountered. Then successful forgery is equivalent to finding a k-coloring of Γ.*

Proof. Suppose Eve wants to impersonate Alice. To that effect, she interferes with the commitment step by sending her own commitment Γ_1' to Bob. Since she should be prepared to respond to the challenge $c = 0$, she should know an isomorphism ψ' between Γ and Γ_1'. On the other hand, since she should be prepared for the challenge $c = 1$, she should be able to produce a k-coloring of Γ_1'. Since she knows ψ' and since ψ' is invertible, this implies that she can produce a k-coloring of Γ. This completes the proof in one direction.

The other direction is trivial. $\square$

7. Endomorphisms of groups or rings (Protocol I)

In this section, we describe a realization of the Protocol I from Section 3 based on an algebraic problem known as the *endomorphism problem*, which can be formulated as follows. Given a group (or a semigroup, or a ring, or whatever) G and two elements $g, h \in G$, find out whether or not there is an endomorphism of G (i.e., a homomorphism of G into itself) that takes g to h.

For some particular groups (and rings), the endomorphism problem is known to be equivalent to the Diophantine problem (see [12, 13]), and therefore the decision problem in these groups is algorithmically unsolvable [9], which implies that the related search problem does not admit a solution in time bounded by any recursive function of the size of an input.

We also note at this point that there is evidence (see e.g. [14]) that finding an isomorphism (or a nontrivial homomorphism) between (finite-dimensional) *algebras* over $\mathbf{Q}$ is hard.

Below we give a description of the authentication protocol based on the endomorphism problem, without specifying a platform group (or a ring), and then discuss possible platforms.

1. Alice's public key consists of a group (or a ring) G and two elements $g, h \in G$, such that $\varphi(g) = h$ for some endomorphism $\varphi \in End(G)$. This φ is Alice's private key.
2. To begin authentication, Alice selects an automorphism ψ of G and sends the element $v = \psi(h)$ (the commitment) to Bob.
3. Bob chooses a random bit c and sends it to Alice.

- If $c = 0$, then Alice sends the automorphism ψ to Bob, and Bob checks whether $v = \psi(h)$ and whether ψ is an automorphism.
- If $c = 1$, then Alice sends the composite endomorphism $\psi\varphi = \psi(\varphi)$ to Bob, and Bob checks whether $\psi\varphi(g) = v$ and whether $\psi\varphi$ is an endomorphism.

Here we point out that checking whether a given map is an endomorphism (or an automorphism) depends on how the platform group G is given. If, for example, G is given by generators and defining relators, then checking whether a given map is an endomorphism of G amounts to checking whether every defining relator is taken by this map to an element equal to 1 in G. Thus, the *word problem* in G (see e.g. [8] or [11]) has to be efficiently solvable.

Checking whether a given map is an automorphism is more complex, and there is no general recipe for doing that, although for a particular platform group that we describe in subsection 7.1 this can be done very efficiently. In general, it would make sense for Alice to supply a proof (at the response step) that her ψ is an automorphism; this proof would then depend on an algorithm Alice used to produce ψ.

Proposition 5. *Suppose that after several runs of steps (2)-(3) of the above protocol, both values of c are encountered. Then successful forgery is equivalent to finding an endomorphism φ such that $\varphi(g) = h$, and is therefore NP-hard in some groups (and rings) G.*

The proof is similar to that of Proposition 4. We also note that in [7], a class of rings is designed for which the problem of existence of an endomorphism between two given rings from this class is NP-hard.

A particular example of a group with the NP-hard endomorphism problem is given in the following subsection.

7.1. Platform: free metabelian group of rank 2

A group G is called *abelian* (or commutative) if $[a, b] = 1$ for any $a, b \in G$, where $[a, b]$ is the notation for $a^{-1}b^{-1}ab$. This can be generalized in different ways. A group G is called *metabelian* if $[[x, y], [z, t]] = 1$ for any $x, y, z, t \in G$. The commutator subgroup of G is the group $G' = [G, G]$ generated by all commutators, i.e., by expressions of the form $[u, v] = u^{-1}v^{-1}uv$, where $u, v \in G$. The second commutator subgroup G'' is the commutator of the commutator of G.

Definition 1. Let F_n be the free group of rank n. The factor group F_n/F_n'' is called the *free metabelian group* of rank n, which we denote by M_n.

Roman'kov [13] showed that, given any Diophantine equation E, one can efficiently (in linear time in the "length" of E) construct a pair of elements u, v of the group M_2, such that to any solution of the equation E, there corresponds an endomorphism of M_2 that takes u to v, and vice versa. Therefore, there are pairs of elements of M_2 for which the endomorphism problem is NP-hard (see e.g. [4, Problem AN8]). Thus, if a free metabelian group is used as the platform for the protocol in this section, then, by Proposition 5, forgery in that protocol is NP-hard.

7.2. Platform: $\mathbf{Z}_p^*$

Here the platform group is $\mathbf{Z}_p^*$, for a prime p. Then, since $\mathbf{Z}_{p-1}^*$ acts on $\mathbf{Z}_p^*$ by automorphisms, via the exponentiation, this can be used as the platform for the Protocol II. In this case, forgery is equivalent to solving the discrete logarithm problem, by Proposition 5.

Acknowledgement. The first author is grateful to Max Planck Institut für Mathematik, Bonn for its hospitality during the work on this paper.

References

[1] P. Caballero-Gil and C. Hernández-Goya, *Strong Solutions to the Identification Problem*, in: 7th Annual International Conference COCOON 2001, Lecture Notes Comp. Sc. **2108** (2001), 257–262.

[2] P. Caballero-Gil and C. Hernández-Goya, *A Zero-Knowledge Identification Scheme Based on an Average-Case NP-Complete Problem*, in: Computer Network Security, MMM-ACNS 2003, St. Petersburg, Russia. Lecture Notes Comp. Sc. **2776** (2003), 289–297.

[3] U. Feige, A. Fiat and A. Shamir, *Zero knowledge proofs of identity,* Journal of Cryptology **1** (1987), 77–94.

[4] M. Garey, J. Johnson, *Computers and Intractability, A Guide to NP-Completeness*, W. H. Freeman, 1979.

[5] O. Goldreich, *Foundations of cryptography*, Cambridge University Press, 2001.

[6] O. Golderich, 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.

[7] D. Grigoriev, *On the complexity of the "wild" matrix problems, of the isomorphism of algebras and graphs*, Notes of Scientific Seminars of LOMI **105** (1981), 10–17 (in Russian) [English translation in J. Soviet Math. **22** (1983), 1285–1289].

[8] R. C. Lyndon, P. E. Schupp, *Combinatorial Group Theory*, Ergebnisse der Mathematik, band 89, Springer 1977. Reprinted in the Springer Classics in Mathematics series, 2000.

[9] Yu. Matiyasevich, *Hilbert's 10th Problem (Foundations of Computing)*, The MIT Press, 1993.

[10] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC-Press 1996.

[11] A. G. Myasnikov, V. Shpilrain, and A. Ushakov, *Group-based cryptography*, Birkhäuser 2008.

[12] V. A. Roman'kov, *Unsolvability of the problem of endomorphic reducibility in free nilpotent groups and in free rings*, Algebra and Logic **16** (1977), 310-320.

[13] V. A. Roman'kov, *Equations in free metabelian groups*, Siberian Math. J. **20** (1979), 469-471.

[14] L. Rónyai, *Simple Algebras Are Difficult*, Proceedings of the Annual ACM Symposium on Theory of Computing (1987), 398–408.

[15] A. Seress, *Permutation Group Algorithms*, Cambridge University Press, 2002.

[16] R. Venkatesan, L. Levin, *Random Instances of a Graph Coloring Problem are Hard*, Proceedings of the Annual ACM Symposium on Theory of Computing (1988), 217–222.