How Traveling Salespersons Prove Their Identity

Stefan Lucks

Institut für Numerische und Angewandte Mathematik
Georg–August–Universität Göttingen
Lotzestr. 16–18, D–37083 Göttingen, Germany
(email: lucks@namu01.gwdg.de)

Abstract. In this paper a new identification protocol is proposed. Its security is based on the *Exact Traveling Salesperson Problem* (XTSP). The XTSP is a close relative of the famous TSP and consists of finding a Hamiltonian circuit of a given length, given a complete directed graph and the distances between all vertices. Thus, the set of tools for use in public-key cryptography is enlarged.

1 Introduction

In public-key cryptography it is common to base the security of a cryptosystem on the hardness of number theoretical problems. This remains true for zero-knowledge identification schemes. Motivations to consider other problems are:

- Cryptosystems based on number theory tend to be only moderately efficient, since they typically depend on multiplying large numbers.
- It is dangerous to have all eggs in one basket, i.e. to depend completely on the same source of problems.

In 1989 Shamir [10] published an identification scheme based on an NP-hard algebraic problem, the *Permuted Kernel Problem*. For results about its hardness see [8] and references therein. Two more recent approaches came from Stern. He considered a coding problem, *Syndrome Decoding* [11] and a combinatorial problem, *Constrained Linear Equations* [12]. Both are NP-hard. This year, Pointcheval [9] proposed another identification scheme, based on the NP-hard *permuted Perceptrons Problem* from the theory of learning machines.

All four identification schemes are provably secure—if it is infeasible to solve the underlying problem.

Such an identification scheme's security is lost if the underlying problem is tractable. This motivates us to consider the *Traveling Salesperson Problem* (TSP), one of the oldest and most prominent problems in algorithm and computational complexity theory. It is NP-hard, but has been studied long before the theory of NP-hardness was developed, see [4], chapter 1.

This paper introduces a new scheme, based on the XTSP (*Exact TSP*), a close relative of the TSP where one looks for a Hamiltonian circuit of a given length—instead of the shortest one. Recently this problem was considered for secret-key cryptography, and constructions for one-way hash functions and pseudorandom bit generators based on the XTSP were given [5].

2 The Underlying Problem

In the following $A = (a_{i,j})$ is an $n \times n$-matrix with $a_{i,i} = 0$. By "randomly chosen" we always mean "chosen at random according to the uniform probability distribution". For $i \neq j$, $a_{i,j}$ is randomly chosen from $\{0, \ldots, 2^{l(n)} - 1\}$. We think of A as the distance matrix for distances in the complete directed Graph G_n with n vertices. Therefore the XTSP is actually a family of problems depending on the parameter $l(n)$.

In this paper we only deal with asymmetric XTSPs (i.e. in general $a_{i,j} \neq a_{j,i}$), but our scheme can easily be adopted to symmetric XTSPs.

Any Hamiltonian cycle X for G_n can be coded as an integer with

$$\lceil \log_2((n-1)!) \rceil \quad \text{Bits.}$$

By $\text{Length}_A(X)$ we mean the length of X with respect to A. Given a number B, the XTSP is to find a Hamiltonian cycle X with

$$\text{Length}_A(X) = B.$$

Choosing $l(n) \approx \log_2((n-1)!)$ leads to at least as hard XTSPs as any other choice for $l(n)$ (see [5]), so $l(n) \approx \log_2((n-1)!)$ will be used for our identification scheme.

It is easy to prove the NP-hardness of the XTSP and the NP-completeness of the corresponding existence problem. But since the theory of NP-completeness only deals with worst-case complexity, we need a stronger assumption than simply NP$\neq$P.

Let $l(n) \approx \log_2((n-1)!)$. Given a $n \times n$ distance matrix $A = (a_{i,j})$ with randomly chosen elements $a_{i,j} \in \{0, \ldots, 2^{l(n)}\}$ and the length B of a randomly chosen secret Hamiltoninan cycle X, the *XTSP intractability assumption* states that it is infeasible to find a Y such that $\text{Length}_A(Y)$ holds with non negligible probability.

As usual, a problem is said to be *infeasible*, if there is no efficient algorithm to solve it. We say, a (possibly probabilistic) algorithm is *efficient*, if it computes a correct solution in polynomial time (in n) with *non negligible* probability, i.e. with a probability bounded from below by $1/P(n)$ for a polynomial P.

Note that we could as well restrict ourselves to the modular XTSP, where the length of a Hamiltonian cycle is measured modulo $2^{l(n)}$, cf. [5].

3 The Identification Scheme

Alice wants to prove her identity. To archive this, Alice runs through our protocol to demonstrate her possession of a secret knowledge. Our scheme is zero-knowledge, i.e. the verifier learns nothing about the secret, except that the prover is in its possession.

We need a collision resistant hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^l$, where given a hash output h, it is infeasible to compute statistical information about a hash input x with $h = H(x)$. In the following we assume that H has these properties—in addition to the XTSP intractability assumption. Shamir [10] and Stern [11], [12] make similar use of a hash function.

If the hash function H were not collision resistant, our scheme would be vulnerable to attacks similar to those described in [2]. Shamir [10] and Stern [11] were not aware of that problem.

Let π be a permutation over $\{1,\ldots,n\}$. (W. l. o. g. we may as well define π to be a permutation over $\{2,\ldots,n\}$ and set $\pi(1) = 1$.) Then $\pi(A)$ corresponds to the distance matrix where the vertices are permuted according to π, i.e. $\pi(A) = \pi((a_{i,j})) = (a_{\pi(i)\pi(j)})$. Similarly we define $\pi(X)$ for a Hamiltonian cycle X, thus

$$\text{Length}_{\pi(A)}(\pi(X)) = \text{Length}_A(X).$$

Given two distance matrixes $A = (a_{i,j})$ and $A' = (a'_{i,j})$, their sum $A + A' = (a''_{i,j})$ is defined component-wise, i.e. $a''_{i,j} = a_{i,j} + a'_{i,j}$.

Let the matrix A and the length B be known to the public. Alice's secret is a Hamiltonian cycle X with $\text{Length}_A(X) = B$. Alice proves her identity to the verifyer Bob as follows.

1. Alice chooses at random a matrix A^* and a permutation π over $\{1,\ldots,n\}$. She computes $B^* = \text{Length}_{A^*}(X')$, $h_0 = H(A^*)$,
   $X' = \pi(X)$, $h_1 = H(X')$,
   $A' = \pi(A)$ and $h_2 = H(A^* + A')$
   and sends B^*, h_0, h_1 and h_2 to Bob.
2. Bob chooses at random $c \in \{0,1,2\}$ and sends c to Alice.
3. In the case Alice responds
   $c = 0$: A^* and X',
   $c = 1$: $A^* + A'$ and X',
   $c = 2$: A^* and π
4. For Bob, Alice's response is a pair (M, N) resp. (M, σ), where M is a matrix, N a Hamiltonian cycle and σ a permutation over $\{1\ldots,n\}$.
   In the case Bob checks
   $c = 0$: $H(M) = h_0$, $H(X') = h_1$ and $\text{Length}_M(N) = B^*$.
   $c = 1$: $H(M) = h_2$, $H(X') = h_1$ and $\text{Length}_M(N) = B + B^*$.
   $c = 2$: $H(M) = h_1$ and $H(M + \sigma(A)) = h_2$.

We could imagine the following variant of the above scheme. Assume we know N users $U_1, \ldots, U_N$. Let B_i be an identification string of length $l(n)$ for the user A_i. B_i might be generated as collision resistant hash of U_i's name, address and similar information. For every user U we choose a random Hamiltonian circuit X_i and *then* we generate a matrix A such that for all $i \in \{1, \ldots, N\}$ $\text{Length}_A(X_i) = B_i$.

This way, we have made the above scheme identity-based. But it is identity-based in a limited sense, since we can't add new users to the system—except by providing each new user U_j ($j \leq N$) with a pseudonym P_j and the corresponding keys $B_j = H(P_j)$ and X_j—.

4 How Secure is the Scheme?

It is easy to see that Alice (and everyone else who knows X) will always pass the test. What, if Carla, without knowing X, wants to mask herself as Alice?

Due to the collision resistance of H, Carla has to fix the matrices A^*, $A^* + A'$ and the Hamiltonian cycle X' in the first step.

If Carla wants to be able to give the correct reply in the case $c = 2$, she must know a permutation π with $\pi(A) = A' = (A^* + A') - A^*$. (We anticipate that the $a_{i,j}$ are pairwise distinct—then π is unique. Argumenting in a birthday paradox style, we see that for $l(n) \approx \log_2((n-1)!)$, $n(n-1)$ independent random values $a_{i,j} \in \{0, \ldots, 2^{l(n)} - 1\}$ and except for very small n the chance to find two pairs $(i,j) \neq (i',j')$ with $a_{i,j} = a_{i',j'}$ is negligible.)

If $\text{Length}_{A^*}(X') \neq B^*$, Carla is lost in the case $c = 0$. For X' the condition

$$\text{Length}_{A'+A^*}(X') = \text{Length}_{A'}(X') + \text{Length}_{A^*}(X') = B + B^*$$

must hold, else Carla is lost in the case $c = 1$.

Even if $X' \neq \pi(X)$, we have

$$\text{Length}_{A'}(X') = B = \text{Length}_A(\pi^{-1}(X')).$$

Thus, if Carla is able to give the correct answer in the cases $c = 0$, $c = 1$ *and* $c = 2$, either she can solve the XTSP or break the hash function H.

If our assumptions hold, the probability for Carla to successfully mask herself as Alice is at most $2/3$. If the identification procedure is repeated r times and Bob's challenges c are chosen independently at random, Carla's chances to remain undetected are bounded by $(2/3)^r$.

What about our claim that the scheme is zero-knowledge? I.e. what does Bob learn about the secret X, if Alice honestly proves her identity to him?

Given h_0, h_1 and h_2, Bob can't compute statistical information about A^*, X' or $A^* + A'$. The only things which come out of the protocol are a random matrix M and either a random Hamiltonian cycle X' or a random permutation π. B^* is the length of X', or the length of a randomly chosen Hamiltonian cycle if X' is unknown. Thus if Bob chooses the challenge c *before* the first step, he can simulate the protocol without the help of Alice, and the probability distributions of the generated random values are the same with or without Alice's cooperation.

Only the ability to give the expected answer for all possible c is genuine for Alice.

5 The Performance of the Scheme

We anticipate that the matrices A and A^* are generated pseudorandomly. (Note that in the case of the identity-based variant A can't be generated pseudorandomly since A depends on the users' identities.) E.g. let $E : \{0,1\}^k \times \{0,1\}^{l(n)} \longrightarrow \{0,1\}^{l(n)}$ be a (secret key) block cipher encryption function. The key size k is a security parameter. We choose a secret key K_A for the matrix A and generate the $a_{i,j}$ by

$$a_{i,j} = E\big(K_A,\ (i-1) + n(j-1)\big).$$

Here $2^{l(n)} \geq n^2$ and "$(i-1)+n(j-1)$" is treated as $l(n)$-bit string.

Similarly the matrix A^* can be generated depending on a secret key K_{A^*}. In other words, we only need $2k$ bits storage for K_A and K_{A^*} to compute all values of A, $A' = \pi(A)$, A^* and $A^* + A'$ efficiently on demand. The complete identification scheme can be implemented using small storage space.

The same idea helps to limit communication requirements. In two of three rounds (on the average), Alice sends the matrix A^* to Bob. It saves bandwidth if Alice sends the key K_{A^*} and Bob generates the values $a^*_{i,j}$ on his own.

It is not that easy to save bandwidth in the case $c = 1$, where Alice sends the matrix $A^* + A'$. It seems impossible to enable Bob to generate the values of that matrix on his own without leaking information about A^*.

Our solution is based on the simple observation that in order to check $\text{Length}_{A^*+A'}(X')$, Bob only needs to know the n distances defined by X'. On the other hand it would be easy for Carla to cheat, if Bob couldn't verify that the distances he sees are the same as the corresponding values of a matrix M with $h_2 = H(M)$.

The following construction is similar to the parallelized version of Damgård's hash function [1] and Merkle's tree authentication [6]. To the author's knowledge, it is the first use of tree authentication in an identification scheme.

The basic idea is to use the given hash function H for small inputs only. Let $(x_1, \ldots, x_n)$ with $x_i \in \{0,1\}^{l(n)}$ be a large input; instead of $H(x_1, \ldots, x_n)$ we compute $H^*(x_1, \ldots, x_n)$ with

$$H^*(x_i, \ldots, x_j) = H\left(H^*\left(x_i, \ldots, x_{i-1+\lceil (j-i)/2 \rceil}\right), H^*\left(x_{i+\lceil (j-i)/2 \rceil}, \ldots, x_j\right)\right)$$

for $i < j$ and $H^*(x_i) = H(x_i)$. We regard this as a binary computation tree with $H^*(x_1, \ldots, x_m)$ as root and $x_1, \ldots, x_m$ as leafs.

Let $h = H^*(x_1, \ldots, x_m)$ and x be public knowledge. Given $x \in \{0,1\}^{l(n)}$, in order to prove $x = x_i$ we only need to reveal i and at most $\lceil \log_2 m \rceil$ intermediate results $H^*(\ldots)$, following the tree from the leaf x_i to the root.

Our case is slightly more complicated. For simplicity we write $a^+_{i,j} = a^*_{i,j} + a'_{i,j}$. We modify our scheme such that no longer $h_2 = H(A^* + A')$, but

$$h_2 = H\left(\ H^*\left(a^+_{1,1}, \ldots, a^+_{1,n}\right),\ \ldots,\ H^*\left(a^+_{n,1}, \ldots, a^+_{n,n}\right)\ \right).$$

(The values $a^+_{i,i}$ could be left out and should be 0 by definition. Here, their use helps to keep the above expression simple.)

Alice has to reveal n values $a^+_{i,j}$. More precise, she has to reveal exactly one value a^+_{1,j_1}, exactly one value a^+_{2,j_2}, ... and exactly one value a^+_{n,j_n}.To prove the correctness of a^+_{i,j_i} she reveals about $\log_2(n)$ intermediate results—instead of $n-1$ leafs.

When computing h_2, Alice knows in advance, which values $a^+_{i,j}$ to reveal if Bob sends her $c = 1$. In order to limit her computing load, Alice can store the necessary intermediate results.

If storage space is short, Alice can resort to a time-space tradeoff. She only stores intermediate results, which are close to the root and thus require a high effort to be computed again. The remaining intermediate results actually have to be computed again—if Bob's challenge for Alice is $c = 1$.

The heaviest part of Alice's computing load is to compute the hashes h_0 and h_2. Since Alice's computations may actually take place on a portable device with limited computation power, hashing large distance matrices may be troublesome. But note that these hashes do not depend on what Bob does, i.e. can be computed during the idle time of Alice's device.

6 The Security Parameter n

It is difficult to recommend a minimum size for n. A complete enumeration of all Hamiltonian circuits will solve the XTSP in $(n-1)!$ steps—but what about more efficient algorithms?

There are good heuristics for the "normal" TSP, especially if the triangle inequality holds. Padberg and Rinaldi [7] solved TSPs with $n =$ "several thousand". But all successful approaches on large-scale TSPs are based on branch–and–bound or (as in the case of Padberg and Rinaldi) on branch–and–cut. The basic branch–and–... principle can roughly be described as follows:

- Start with solution space S, and divide it into subspaces $S_1, S_2, \ldots, S_k$.
- Compute lower and/or upper bounds for all solutions in S_i.
- For all S_i do:
  - If the lower bound is too large (or the upper bound too low) discard S_i
  - else branch, i.e. apply branch–and–... on the solution space S_i.

In the case of the XTSP, this leads to excessive branching because the solution space S we start with (a set of $(n-1)!$ Hamiltonian cycles) and most subspaces can be expected to contain both too long and too short Hamiltonian cycles.

Empiric evidence supports this reasoning. Thienel [13] wrote a TSP-solver for TSPs under constraints, based on the TSP-code of [3]. He also made some experiments with XTSPs. The condition that the solution is a Hamiltonian circuit with a fixed length can be viewed as such a constraint.

The program was run on a Sun Sparcstation 10 with 60 Mhz and ran out of memory when more than about 30000 branch–and–bound nodes had to be stored. It was not designed for such large branch–and–bound trees. Also it was written for symmetric TSPs. The following distance matrices were considered:

- `gr17`, a symmetric 17*17 matrix from the TSPLIB and
- `pi`x`-1E`y, symmetric $x \times x$ matrices with random distances $\in \{0, \ldots, 10^y - 1\}$ (derived from the digits of π) and

$$(x, y) \in \{(13, 8), (14, 8), (18, 3), (18, 4), (18, 5), (18, 8)\}.$$

For every distance matrix, five random Hamiltonian circuits were chosen and their lengths computed. Among these 35 XTSPs, all `gr17`-based, four `pi-18E3`-based and two `pi-18E4`-based problems were solved, the remaining ones could not be solved due to lack of memory. The required resources can be found in table 1; the time is given in [min:sec]. The corresponding unconstrained TSPs were easy to solve, requiring one branch-and-bound node and < 1 sec time.

Table 1.

matrix	nodes	time
`gr17`	323	0:13
	333	0:14
	1575	1:06
	1658	0:47
	1193	0:47
`pi18-1E3`	6881	8:05
	2933	3:00
	5515	5:49
	1575	1:43
	1 unsolved	
`pi18-1E4`	4361	4:29
	157	0:07
	3 unsolved	
`pi18-1E5`	5 unsolved	
`pi13-1E8`	5 unsolved	
`pi14-1E8`	5 unsolved	
`pi18-1E8`	5 unsolved	

XTSPs based on `pi18-1E3`, `pi18-1E4` and `pi18-1E5` are underconstrained, i.e. in general there is more than one Hamiltonain circuit with a given length.

As could be expected, the program's success rate drops fast when the number length increases.

Even non-heuristic algorithms for the "normal" TSP like the $O(n^2 2^n)$ dynamic programming algorithm (see [4], chapter 3) don't seem to be adaptable to the XTSP.

It appears reasonable to estimate that XTSPs with $n = 21$ are hard, but can be solved with present technology ($20! \approx 2^{61}$), while XTSPs with $n = 41$ can not. When $n = 41$, we have to deal with 160-bit numbers since $l(n) = 160 \approx \log_2(40!)$.

It is not recommended to adopt a cryptographic scheme for actual use too early. This remains true for the scheme proposed here. Experts in cryptanalysis and/or combinatorics are invited to search for efficient algorithms to solve the XTSP.

Acknowledgements

I am grateful to an anonymous referee of [5] for pointing me towards identification schemes and to Stefan Thienel for providing the empiric results.

References

1. I. B. Damgård, *A Design Principle for Hash Functions*, in: Proc Crypto '89, Springer LNCS 435, 416–427.
2. M. Girault, J. Stern, *On the length of cryptographic hash-values used in identification schemes*, in: Proc. Crypto '94, Springer LNCS 839, 202-215.
3. M. Jünger, G. Reinelt, S. Thienel, *Provably good solutions for the traveling salesman problem*, in: Zeitschrift für Operations Research 40 (1994), 183–217.
4. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, D. B. Shmoys (eds.), *The Traveling Salesman Problem*, Wiley, 1985.
5. S. Lucks, *How to Exploit the Intractability of Exact TSP for Cryptography*, to appear in: Proc. Fast Software Encryption (1994), Springer LNCS.
6. R. C. Merkle, *A Certified Digital Signature (That Antique Paper from 1979)*, in: Crypto '89, Springer LNCS 435, 218–238.
7. M. Padberg, G. Rinaldi, *A Branch–and–Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems*, in: Siam Review 33, No. 1 (1991), 60-100.
8. J. Patarin, P. Chauvaud, *Improved Algorithms for the Permuted Kernel Problem*, in: Proc. Crypto '93, Springer LNCS 773, 391-402.
9. D. Pointcheval, *A New Identification Scheme Based on the Perceptrons Problem*, in: Proc. Eurocrypt '95, Springer LNCS 921, 319–328.
10. A. Shamir, *An Identification Scheme based on Permuted Kernels*, in: Proc. Crypto '89, Springer LNCS 435, 606-609.
11. J. Stern, *A new identification scheme based on syndrome decoding*, in: Proc. Crypto '93, Springer LNCS 773, 13-20.
12. J. Stern, *Designing Identification Schemes with Keys of Short Size*, in: Proc. Crypto '94, Springer LNCS 839, 164-173.
13. S. Thienel, private communication, July 1995.

This paper was published at:
Fifth IMA Conference on Cryptography
and Coding, Springer LNCS 1025, 142–149.