Noisy Subset Product

Trey Li

The University of Manchester, Manchester, UK
trey.li@manchester.ac.uk

Abstract. In 1978, Yao studied the subset product problem and proved its NP-completeness. Later, Galbraith, Zobernig, and Li considered a prime-based modular variant and studied its average-case hardness. More recently, Li introduced the general problem of unknown-coefficient multivariate exponential system solving and studied its hardness systematically in an abstract setting. Li's framework implies a noisy modular variant of subset product as a special case. In this paper, we study this noisy subset product problem in a concrete setting, prove equivalence relations among several related variants, and use these results to construct a post-quantum non-alternating key exchange scheme, with a natural one-sided interpretation as a key encapsulation mechanism. Unlike other noise-based schemes, such as lattice-based schemes, our construction does not rely on an additional reconciliation plaintext. Instead, the receiver is able to recover the sender's ephemeral secret key itself. Moreover, we reduce indistinguishability security directly to the search version of the noisy subset product problem, without relying on a separate decisional assumption.

Keywords: Noisy subset product, unknown-coefficient multivariate exponential equations, post-quantum cryptography, non-alternating key exchange, key encapsulation mechanism

1 Introduction

1.1 Noisy subset product

The subset product problem [16, p.224] is a classical NP-complete problem. It is about solving the multivariable exponential equation

$$\prod_{i=1}^{n} a_i^{x_i} = b$$

for a binary solution $(x_1, \ldots, x_n) \in \{0,1\}^n$, where $a_1, \ldots, a_n, b \in \mathbb{Z}$. Andrew Yao showed its NP-completeness in a private communication in 1978 [16, p.224, p.325]. However, due to the lack of applications, the problem has not gained much attention in the last several decades.

From 2019 to 2021, Galbraith, Zobernig, and Li studied the *modular subset product problem* with prime bases $a_1, \ldots, a_n$ and found cryptographic applications [15, 14]. More recently, Li introduced the general problem of *unknown-coefficient multivariate exponential system solving* (MESS) [21], which implies a noisy version of the modular subset product problem as a special case. In this paper, we study this noisy variant, which we call the *noisy subset product problem*. The problem is about solving a system of noisy exponential equations of the form

$$\prod_{i=1}^{n} a_i^{x_i} \cdot e \equiv b \pmod{q}$$

for a common binary solution $(x_1, \ldots, x_n) \in \{0,1\}^n$, where each a is a random integer, each e is a random prime or the reciprocal of a random prime, each b is an integer computed from the independently random a's and e as well as a common random vector $(x_1, \ldots, x_n) \in \{0,1\}^n$ and a common prime modulus q. The a's, b's and q are given, while the x's and e's are unknowns.

More concretely, let $L = \{\ell_1, \ldots, \ell_k\}$ be a set of primes and q be a prime modulus that is greater than the product of any $2n+1$ primes in L. Let $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ be a random binary vector. A noisy subset product instance with respect to $\boldsymbol{x}$ is a tuple $(a_1, \ldots, a_n, b)$, where $a_1, \ldots, a_n$ are uniform numbers in $\mathbb{Z}_q^\times$, $b := \prod_{i=1}^{n} a_i^{x_i} \cdot e \pmod{q}$, and e is a random prime sampled from L. The noisy subset product problem

is given oracle access to this kind of instances with respect to the same $\boldsymbol{x}$ sampled uniformly from $\{0,1\}^n$, and asks to find $\boldsymbol{x}$. In plain language, the noisy subset product problem is just the modular version of the classical subset product problem [16, p.224] employing noises so that the solution $\boldsymbol{x}$ can be reused in many random instances without trivializing the problem. We conjecture that this problem is hard to solve for both classical and quantum computers.

1.2 Key exchange idea

We explain how noisy subset product leads to a key-exchange scheme.

Let $\boldsymbol{a} = (a_1, \ldots, a_n)$ be a vector of integers and $\boldsymbol{x} = (x_1, \ldots, x_n)$ be a vector of bits. Define $\boldsymbol{a}^{\boldsymbol{x}} = {}^{\boldsymbol{x}}\boldsymbol{a} := \prod_{i=1}^{n} a_i^{x_i}$. Let $\boldsymbol{A}$ be a square matrix of integers with n rows $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n$ and n columns $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n$. Define $\boldsymbol{A}^{\boldsymbol{x}} := (\boldsymbol{r}_1^{\boldsymbol{x}}, \ldots, \boldsymbol{r}_n^{\boldsymbol{x}})$ and ${}^{\boldsymbol{x}}\boldsymbol{A} := ({}^{\boldsymbol{x}}\boldsymbol{c}_1, \ldots, {}^{\boldsymbol{x}}\boldsymbol{c}_n)$. They are simply subset products with respect to the rows and columns of $\boldsymbol{A}$ respectively. Let $\boldsymbol{e} = (e_1, \ldots, e_n)$ be a vector of primes. Define $\boldsymbol{A}^{\boldsymbol{x}} \circ \boldsymbol{e} := (\boldsymbol{r}_1^{\boldsymbol{x}} \cdot e_1, \ldots, \boldsymbol{r}_n^{\boldsymbol{x}} \cdot e_n)$ and ${}^{\boldsymbol{x}}\boldsymbol{A} \circ \boldsymbol{e} := ({}^{\boldsymbol{x}}\boldsymbol{c}_1 \cdot e_1, \ldots, {}^{\boldsymbol{x}}\boldsymbol{c}_n \cdot e_n)$. They are simply noisy subset products with respect to the rows and columns of $\boldsymbol{A}$ respectively. We have the following associativity:

$$ {}^{\boldsymbol{y}}(\boldsymbol{A}^{\boldsymbol{x}}) = {}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{x}} = ({}^{\boldsymbol{y}}\boldsymbol{A})^{\boldsymbol{x}}. $$

This is simply the product of the elements in $\boldsymbol{A}$ that are simultaneously "chosen" by $\boldsymbol{x}$ and $\boldsymbol{y}$. See Fig. 1 for an example.

	0	1	1	0	0
0	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
0	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
1	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$
1	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$
0	$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$

Fig. 1. Example of the associativity ${}^{\boldsymbol{y}}(\boldsymbol{A}^{\boldsymbol{x}}) = {}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{x}} = ({}^{\boldsymbol{y}}\boldsymbol{A})^{\boldsymbol{x}}$, where $\boldsymbol{x} = (0,1,1,0,0)$ and $\boldsymbol{y} = (0,0,1,1,0)$. The number ${}^{\boldsymbol{y}}(\boldsymbol{A}^{\boldsymbol{x}})$ is the product of elements of $\boldsymbol{A}$ that are "chosen" firstly by $\boldsymbol{x}$ then by $\boldsymbol{y}$. The number $({}^{\boldsymbol{y}}\boldsymbol{A})^{\boldsymbol{x}}$ is the product of elements of $\boldsymbol{A}$ that are "chosen" firstly by $\boldsymbol{y}$ then by $\boldsymbol{x}$. They are equally the product of the "intersecting" elements $a_{3,2}$, $a_{3,3}$, $a_{4,2}$ and $a_{4,3}$.

A Diffie-Hellman-like diagram for this associativity is the following

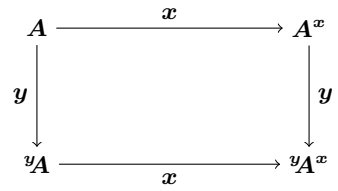

Fig. 2. Diffie-Hellman-like diagram from multiple subset products (insecure).

However, the naive key exchange from this diagram is insecure since without noises the multiple subset product problem is efficiently solvable by linear algebra after taking discrete logarithms or Legendre symbols.

Our dream is the following *noisy Diffie-Hellman* diagram, where the secrets $\boldsymbol{x}$ and $\boldsymbol{y}$ are protected by the noisy subset product problem.

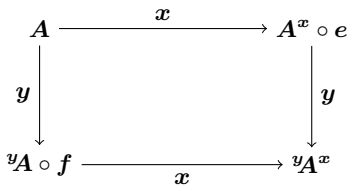

Fig. 3. Noisy Diffie-Hellman diagram.

To achieve this diagram, the bottleneck is how to compute ${}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{x}} \pmod q$ from $(\boldsymbol{A}^{\boldsymbol{x}} \circ \boldsymbol{e} \pmod q, \boldsymbol{y})$ and $({}^{\boldsymbol{y}}\boldsymbol{A} \circ \boldsymbol{f} \pmod q, \boldsymbol{x})$ respectively. A natural idea is to recover $\boldsymbol{x}$ from $\boldsymbol{A}^{\boldsymbol{x}} \circ \boldsymbol{e} \pmod q$ and $\boldsymbol{y}$ from ${}^{\boldsymbol{y}}\boldsymbol{A} \circ \boldsymbol{f} \pmod q$, then the product ${}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{x}} \pmod q$ is immediate. We will show how to achieve this goal using trapdoors.

Importantly, if $\boldsymbol{x}$ and $\boldsymbol{y}$ can be recovered, then there is no need to use ${}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{x}} \pmod q$ as the shared secret. In particular, Alice and Bob may instead set the shared secret to be $(\boldsymbol{a}^{\boldsymbol{x}} \pmod q, \boldsymbol{b}^{\boldsymbol{y}} \pmod q)$ for two extra base vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. Interestingly, this modification allows the "indistinguishability" security of the scheme to be reduced directly to the "search" version of the noisy subset product problem, without relying on a separate Diffie-Hellman-like "decisional" assumption.

Therefore our true scheme is as follows, which is Fig. 3 with ${}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{x}} \pmod q$ replaced with $(\boldsymbol{a}^{\boldsymbol{x}} \pmod q, \boldsymbol{b}^{\boldsymbol{y}} \pmod q)$.

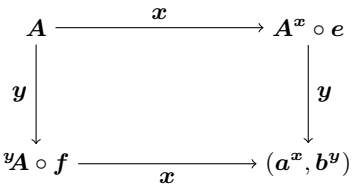

Fig. 4. Our true scheme.

Here we explain the scheme on one side, where Bob sends an ephemeral public key to Alice, and Alice computes the shared secret. The full scheme has another symmetric side from Alice to Bob, which is explained in Section 5.

Let $\boldsymbol{A}$ be an $n \times n$ public matrix of random integers and q be a prime modulus that is greater than the product of any $2n+1$ noise primes. Alice's secret key is a vector $\boldsymbol{s} = (s_1, \ldots, s_n)$ of uniform bits together with a vector $\boldsymbol{u} = (u_1, \ldots, u_n)$ of random noise primes. Her public key is a sequence of noisy subset products $S = (S_1, \ldots, S_n) = \boldsymbol{A}^{\boldsymbol{s}} \circ \boldsymbol{u} \pmod q$. Her ephemeral secret key is a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ of uniform bits together with a vector $\boldsymbol{e} = (e_1, \ldots, e_n)$ of random noise primes. Bob's ephemeral secret key is a vector $\boldsymbol{y} = (y_1, \ldots, y_n)$ of uniform bits together with a vector $\boldsymbol{f} = (f_1, \ldots, f_n)$ of random noise primes and an extra random noise prime f. His ephemeral public key is a sequence of noisy subset products $B = (B_1, \ldots, B_n) \equiv {}^{\boldsymbol{y}}\boldsymbol{A} \circ \boldsymbol{f} \pmod q$. To share $\boldsymbol{y}$ with Alice, Bob computes a *composite* noisy subset product $D = {}^{\boldsymbol{y}}S \cdot f \equiv ({}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{s}} \cdot {}^{\boldsymbol{y}}\boldsymbol{u}) \cdot f \pmod q$ and sends (B, D) to Alice. Alice then computes a composite subset product $B^{\boldsymbol{s}} \equiv {}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{s}} \circ \boldsymbol{f}^{\boldsymbol{s}} \pmod q$ using her secret key $\boldsymbol{s}$ and multiplies $B^{\boldsymbol{s}}$ and D together. Alice then has $B^{\boldsymbol{s}} \cdot D \equiv ({}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{s}} \cdot \boldsymbol{f}^{\boldsymbol{s}}) \cdot (({}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{s}} \cdot {}^{\boldsymbol{y}}\boldsymbol{u}) \cdot f) \equiv {}^{\boldsymbol{y}}\boldsymbol{u} \cdot \boldsymbol{f}^{\boldsymbol{s}} \cdot f \pmod q = \boldsymbol{u}^{\boldsymbol{y}} \cdot \boldsymbol{f}^{\boldsymbol{s}} \cdot f$, where the last step is due to the fact that q is greater than the product of any $2n+1$ noise primes but the Hamming weights of $\boldsymbol{s}$ and $\boldsymbol{y}$ are at most n. Now Alice can recover each bit y_i of $\boldsymbol{y}$ by testing whether u_i divides $B^{\boldsymbol{s}} \cdot D$ (assume that all noise primes in $\boldsymbol{u}$ and $\boldsymbol{f}$ are different). She then computes $(\boldsymbol{a}^{\boldsymbol{x}} \pmod q, \boldsymbol{b}^{\boldsymbol{y}} \pmod q)$ as the shared secret. See Fig. 5 for a concrete example.

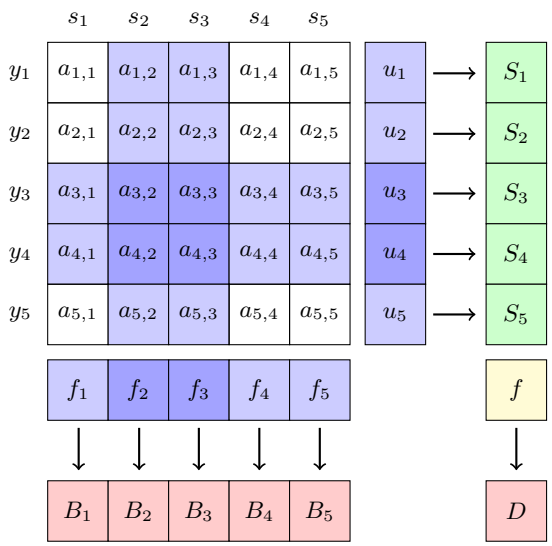

Fig. 5. Example of one side of our key exchange, where $\boldsymbol{s} = (0, 1, 1, 0, 0)$ and $\boldsymbol{y} = (0, 0, 1, 1, 0)$. The composite product contains the "intersecting" elements $a_{3,2}$, $a_{3,3}$, $a_{4,2}$, $a_{4,3}$, u_3, u_4, f_2 and f_3 that are simultaneously "chosen" by $\boldsymbol{s}$ and $\boldsymbol{y}$. After dropping the intersecting a's from the composite product, the remaining product $u_3 \cdot u_4 \cdot f_2 \cdot f_3 \cdot f \bmod q = u_3 \cdot u_4 \cdot f_2 \cdot f_3 \cdot f$ only contains noise primes. Alice can then learn y_i by testing whether $u_i | (u_3 \cdot u_4 \cdot f_2 \cdot f_3 \cdot f)$ using her local noise primes $u_1, \ldots, u_5$. Having $\boldsymbol{y}$, together with her own secret key $\boldsymbol{x}$, Alice can compute the shared secret $(\boldsymbol{a}^{\boldsymbol{x}} \pmod q, \boldsymbol{b}^{\boldsymbol{y}} \pmod q)$ immediately.

Note that the decoding method is very simple. It only uses two basic facts: (1) if $a < q$ then $a \bmod q = a$; and (2) a prime p_i divides a square-free integer $a = \prod_{i=1}^{k} p_i^{x_i}$ if and only if $x_i = 1$.

The security is based only on the hardness of the noisy subset product problem. This can be thought of as the security of classical Diffie-Hellman key exchange relying only on the hardness of the discrete logarithm problem without the Diffie-Hellman assumption. The intuition here is that the additional pairs $(\boldsymbol{a}, \boldsymbol{a}^{\boldsymbol{x}} \pmod q)$ and $(\boldsymbol{b}, \boldsymbol{b}^{\boldsymbol{y}} \pmod q)$ can be viewed as two extra noisy subset product instances in which the noises are fixed to be 1. This leads to the incomplete noisy subset product variant, which is the noisy subset product problem including one noiseless instance. We show that, for a polynomial-size noise set L, the search and decision versions of these two problems are all equivalent. This allows the indistinguishability security of the key exchange scheme to be reduced all the way down to the basic noisy subset product problem.

1.3 Contributions

This paper studies the noisy subset product problem as a special case of the general problem of MESS [21]. Our contributions are twofold.

First, on the problem side, we isolate this concrete noisy modular variant of subset product, formulate it in a way suitable for cryptographic use, and develop several structural properties needed later in the paper. In particular, we study its relation with several related variants, including trapdoor and incomplete variants, and prove equivalence relations that allow cryptographic security to be reduced to the basic search version of the problem.

Second, on the cryptographic side, we construct a post-quantum non-alternating key exchange scheme from noisy subset product, with a natural one-sided interpretation as a key encapsulation mechanism. The construction avoids the use of an additional reconciliation plaintext and instead allows recovery of the ephemeral secret key, which in turn enables a compact multi-party key-exchange extension; see Section 5.5 for details. Moreover, the scheme has a clean security reduction directly to the search version of noisy subset product, without relying on a separate Diffie-Hellman-type assumption.

1.4 Organization

Section 2 defines notation. Section 3 introduces the noisy subset product problem and studies its hardness. Section 4 develops the intermediate variants and equivalence relations needed for the later security reductions. Section 5 presents our non-alternating key exchange scheme, together with its correctness, efficiency, and security proofs. Section 6 discusses two attacks that apply when the ephemeral secret key is reused. Appendix A sketches further possible applications of noisy subset product.

2 Notation

We denote matrices by bold uppercase letters such as $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$, etc. We denote vectors by bold lowercase letters such as $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, etc., or sometimes by light uppercase letters such as A, B, C, etc.

Let $\boldsymbol{a} = (a_1, \ldots, a_n)$, $\boldsymbol{b} = (b_1, \ldots, b_n)$, $\boldsymbol{c} = (c_1, \ldots, c_n)$, $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$ be vectors of real numbers, and let x, y and z be real numbers. Define $\boldsymbol{a} \circ \boldsymbol{b} := (a_1 \cdot b_1, \ldots, a_n \cdot b_n)$. We have $(\boldsymbol{a} \circ \boldsymbol{b}) \circ \boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c} = \boldsymbol{a} \circ (\boldsymbol{b} \circ \boldsymbol{c})$ (associativity) and $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{b} \circ \boldsymbol{a}$ (commutativity). Define $\boldsymbol{a}^{\boldsymbol{x}} = {}^{\boldsymbol{x}}\boldsymbol{a} := \prod_{i=1}^{n} a_i^{x_i}$.We have $(\boldsymbol{a} \circ \boldsymbol{b})^{\boldsymbol{x}} = \boldsymbol{a}^{\boldsymbol{x}} \cdot \boldsymbol{b}^{\boldsymbol{x}}$ and ${}^{\boldsymbol{x}}(\boldsymbol{a} \circ \boldsymbol{b}) = {}^{\boldsymbol{x}}\boldsymbol{a} \cdot {}^{\boldsymbol{x}}\boldsymbol{b}$ (distributivity). Define $\boldsymbol{a}^{x} := (a_1^x, \ldots, a_n^x)$. We have $\boldsymbol{a}^{(xy)z} = \boldsymbol{a}^{xyz} = \boldsymbol{a}^{x(yz)}$ (associativity), $(\boldsymbol{a}^x)^y = \boldsymbol{a}^{xy} = (\boldsymbol{a}^y)^x$ (commutativity) and $\boldsymbol{a}^{(x+y)z} = \boldsymbol{a}^{xz+yz}$ (distributivity).

Let $\boldsymbol{A}$ be a matrix of real numbers with rows $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m$ and columns $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n$. Denote $\boldsymbol{A}^{\boldsymbol{x}} := (\boldsymbol{r}_1^{\boldsymbol{x}}, \ldots, \boldsymbol{r}_m^{\boldsymbol{x}})$ and ${}^{\boldsymbol{y}}\boldsymbol{A} := ({}^{\boldsymbol{y}}\boldsymbol{c}_1, \ldots, {}^{\boldsymbol{y}}\boldsymbol{c}_n)$. We have $({}^{\boldsymbol{y}}\boldsymbol{A})^{\boldsymbol{x}} = {}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{x}} = {}^{\boldsymbol{y}}(\boldsymbol{A}^{\boldsymbol{x}})$ (associativity).

Let $\boldsymbol{v} = (v_1, \ldots, v_n)$ be a vector of integers and a be an integer. We denote by $\boldsymbol{v}|a \in \{0,1\}^n$ the binary vector such that its i-th entry is 1 if and only if $v_i|a$.

We denote by $s \leftarrow S$ uniform sampling from S if S is a set, or sampling according to S if S is a distribution.

Let P_1 and P_2 be two computational problems. Denote $P_1 \leq P_2$ (called P_1 reduces to P_2) if there exists a probabilistic polynomial time reduction from P_1 to P_2, with respect to the instance distributions specified in their definitions. Denote $P_1 = P_2$ if $P_1 \leq P_2$ and $P_2 \leq P_1$.

3 The noisy subset product problem

3.1 The problem

Recall that the classical subset product problem [16, p.224] is given an integral vector $\boldsymbol{a} \in \mathbb{N}^n$ and an integer $X \in \mathbb{N}$ and asks to find a binary vector $\boldsymbol{x} \in \{0,1\}^n$ such that $\boldsymbol{a}^{\boldsymbol{x}} = X$. The modular subset product problem is given an integral vector $\boldsymbol{a} \in \mathbb{N}^n$, an integer $X \in \mathbb{N}$, and a prime q and asks to find a binary vector $\boldsymbol{x} \in \{0,1\}^n$ such that $\boldsymbol{a}^{\boldsymbol{x}} = X \pmod q$. The multiple modular subset product problem is given multiple instances of modular subset product with respect to the same solution $\boldsymbol{x}$ and asks to find $\boldsymbol{x}$. This problem is trivially solvable if the instances are random and the number of instances is large enough. In fact, one can solve it by firstly taking discrete logarithms or Legendre symbols to get a linear system and then solve the system by linear algebra.

In order to have a hard problem, we introduce noises into each instance and define the *noisy subset product problem* (NSP). Following is the standard, i.e., non-average-case definition of the problem.

Definition 1 (Standard-NSP). *Let L be a set of primes and q be a prime such that $q \notin L$. Let $\boldsymbol{x} \in \{0,1\}^n$ be a binary vector. The noisy subset product problem is given instances of the form $(\boldsymbol{a} \in \mathbb{Z}^n, X \in \mathbb{Z}_q^\times)$ with respect to the same $\boldsymbol{x}$, where $\boldsymbol{a} \in \mathbb{Z}^n$ are vectors of integers, $X := \boldsymbol{a}^{\boldsymbol{x}} \cdot e \pmod q \in \mathbb{Z}_q^\times$ are noisy products, and $e \in L$ are noises, find $\boldsymbol{x}$.*

When the noise set L is small, NSP is NP-complete.

Theorem 1. *If $|L|$ is polynomial in n, then NSP is NP-complete.*

Proof. It suffices to show that NSP with a single instance is NP-complete, as a special case of NSP is that all of its instances are the same. We first show that NSP is in NP. Given an instance $(\boldsymbol{a}, X)$, a certificate consists of a vector $\boldsymbol{x} \in \{0,1\}^n$ together with an element $e \in L$ such that $X \equiv \boldsymbol{a}^{\boldsymbol{x}} \cdot e \pmod{q}$. Since $|L|$ is polynomial in n, the element e can be specified using polynomially many bits, and membership in L can be checked in polynomial time. Moreover, given $(\boldsymbol{x}, e)$, one can compute $\boldsymbol{a}^{\boldsymbol{x}} \cdot e \bmod q$ in polynomial time and verify whether it equals X. Hence NSP is in NP.

For NP-hardness, note that NSP is a special case of MESS over the concrete Euclidean domain $\mathbb{Z}$. Applying [21, Theorem 3.9] together with [21, Lemma 3.8] to this special case gives the desired NP-hardness. This completes the proof.

□

We now give the average-case definition of the problem. This is the main problem this paper studies. For clarity, the parameters are explained in footnotes.

Setup:

Let $L := \{\ell_1, \ldots, \ell_k\}$ be the set of the first k primes.[1]
Let q be the smallest prime greater than ℓ_k^{2n+1}.[2]
Let D_a be the uniform distribution over $\mathbb{Z}_q^\times$.[3]
Let D_L be the distribution that outputs a random prime $e \in L$ with uniform Legendre symbol $\left(\frac{e}{q}\right)$.[4]
Let $D_{\boldsymbol{x}}$ with respect to some $\boldsymbol{x} \in \{0,1\}^n$ be the distribution that samples $\boldsymbol{a} \leftarrow D_a^n$ and $\ell \leftarrow D_L$, sets $e := \ell$, computes $X = \boldsymbol{a}^{\boldsymbol{x}} \cdot e \pmod{q}$, and outputs $(\boldsymbol{a}, X)$.
Let $D'_{\boldsymbol{x}}$ be $D_{\boldsymbol{x}}$ with $e := \ell^{-1}$.
Let $O_{\boldsymbol{x}}$ with $\boldsymbol{x} \leftarrow \{0,1\}^n$ be the oracle that outputs instances $(\boldsymbol{a}, X)$ all sampled from $D_{\boldsymbol{x}}$ or all sampled from $D'_{\boldsymbol{x}}$.
Let O_{ran} be the random oracle that outputs uniform instances $(\boldsymbol{a}, X) \leftarrow (\mathbb{Z}_q^\times)^n \times \mathbb{Z}_q^\times$.

Definition 2 (Search-NSP). *Given access to an NSP oracle $O_{\boldsymbol{x}}$, find $\boldsymbol{x}$.*[5]

Definition 3 (Decision-NSP). *Given $O_{\boldsymbol{x}}$ or O_{ran}, decide which is the case.*

In the rest of the paper, unless specified otherwise or the context is clear, NSP refers to Search-NSP defined in Definition 2.

3.2 Hardness

To gain intuition for why NSP is nontrivial to solve, we analyze several attacks and explain why they fail.

[1] The size of k can be polynomial or superpolynomial in n.

[2] This means that q is greater than the product of any $2n+1$ primes in L. This is for the success decoding of the key exchange scheme. More precisely, we could choose $q \gtrapprox (\ell_k)^{2n+1}$. See the proof of Theorem 9 for more details. Also, we suggest choosing a safe prime, Fermat prime or Mersenne prime for q. The advantage of a safe prime $q = 2p+1$ is that there are only two factors 2 and p of $q-1$, which helps avoid potential attacks that deal with different small subgroups of $\mathbb{Z}_{q-1}$, for example attacks using small power residue symbols. The advantage of a Fermat prime $q = 2^m + 1$ is that $q - 1 = 2^m$ has only one prime factor and thus also avoids attacks that are relevant to dealing with different subgroups of $\mathbb{Z}_{q-1}$ of coprime orders. The advantages of a Mersenne prime are that computing $a \bmod (2^m - 1)$ is easy and that a Mersenne prime of desired size is easy to find. Apart from these, we could also consider pseudo-random primes for easy choice of q.

[3] In practice, we can define D_a as follows. It keeps sampling binary vectors $\alpha = (\alpha_0, \ldots, \alpha_{\lceil \log q \rceil - 1}) \leftarrow \{0,1\}^{\lceil \log q \rceil}$ uniformly at random until finding a nonzero one, and outputs the integer $a := \sum_{i=0}^{\lceil \log q \rceil - 1} (\alpha_i \cdot 2^i)$.

[4] Concretely, it first samples a uniform bit $b \leftarrow \{-1, 1\}$ and then keeps sampling binary vectors $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_{\lceil \log(\ell_k) \rceil - 1}) \leftarrow \{0,1\}^{\lceil \log(\ell_k) \rceil}$ uniformly at random until finding one such that the integer $e := \sum_{i=0}^{\lceil \log(\ell_k) \rceil - 1} (\alpha_i \cdot 2^i)$ is a prime in L and that the Legendre symbol $\left(\frac{e}{q}\right) = b$, and outputs e. This is simply sampling random primes from L until finding one whose Legendre symbol is the uniform b.

[5] Note that finding $\boldsymbol{x}$ implies finding the noises e's. So we only ask to find $\boldsymbol{x}$.

Brute forcing. A naive attack is to brute force the noise term e and remove it from the NSP equations

$$\boldsymbol{a}^{\boldsymbol{x}} \cdot e = X \pmod{q}$$

to get equations of the form

$$\boldsymbol{a}^{\boldsymbol{x}} = X' \pmod{q},$$

where $X' = X \cdot e^{-1} \pmod{q}$. Taking Legendre symbols yields equations of the form

$$\prod_{i=1}^{n} \left(\frac{a_i}{q}\right)^{x_i} = \left(\frac{X'}{q}\right).$$

Let

$$\alpha_i = \frac{1}{2}\left(1 - \left(\frac{a_i}{q}\right)\right) \text{ and } \beta = \frac{1}{2}\left(1 - \left(\frac{X'}{q}\right)\right)$$

we obtain parity equations of the form

$$\sum_{i=1}^{n} \alpha_i x_i = \beta \pmod{2}.$$

For m such equations, we get a parity-check system

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \pmod{2}.$$

By the classical result in [20, 3, 5], the probability that a uniform matrix in $\mathbb{Z}_2^{m\times n}$ with $m > n$ is of full $\mathbb{Z}_2$-rank n is

$$\prod_{i=m-n+1}^{m} \left(1 - \frac{1}{2^i}\right).$$

Hence if $\boldsymbol{A}$ is uniform, then from $m \geq 2n$ NSP instances one has a full-rank $\boldsymbol{A}$ with probability

$$\prod_{i=m-n+1}^{m} \left(1 - \frac{1}{2^i}\right) \geq \prod_{i=n+1}^{2n} \left(1 - \frac{1}{2^i}\right) > \left(1 - \frac{1}{2^n}\right)^n > 1 - \frac{n}{2^n} = 1 - \frac{1}{2^{n-\log_2(n)}},$$

which is overwhelming in n. One can then solve for the unique $\boldsymbol{x}$ by linear algebra. However, this attack requires brute forcing the noises in the m NSP instances, or at least the Legendre symbols of these noises, which is infeasible.

LPN reduction. Following the first attack, now we do not require guessing the noises. Let

$$\varepsilon = \frac{1}{2}\left(1 - \left(\frac{e}{q}\right)\right),$$

which is unknown. Then we have parity equations of the form

$$\sum_{i=1}^{n} \alpha_i x_i + \varepsilon = \beta \pmod{2},$$

and thus a parity-check system

$$\boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon} = \boldsymbol{b} \pmod{2}.$$

This is the learning parity with noise (LPN) problem with a special noise distribution induced by Legendre symbols of random primes. Note that in the definition of NSP, each noise prime is chosen by D_L according to a uniform Legendre symbol, so the resulting noise rate is exactly $1/2$. This means that the resulting LPN instance is information-theoretically unsolvable. Hence standard LPN algorithms such as BKW [7] do not work.

Knapsack reduction. This attack only uses one NSP instance $(\boldsymbol{a}, X)$. When $k = poly(n)$, we append the vector $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_k)$ of all noise primes in L to the end of the base vector $\boldsymbol{a}$ to get a new base vector $\boldsymbol{v} = (\boldsymbol{a}, \boldsymbol{\ell})$. Then the pair $(\boldsymbol{v}, X)$ is a noiseless modular subset product instance, which asks to find a binary vector $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y}) \in \{0,1\}^n \times \{0,1\}^k$ such that

$$\boldsymbol{v}^{\boldsymbol{z}} = \boldsymbol{a}^{\boldsymbol{x}} \boldsymbol{\ell}^{\boldsymbol{y}} = X \pmod{q}.$$

If we take discrete logarithms, the problem is reduced to a subset sum instance $(\log_g(\boldsymbol{v}), \log_g(X))$, which asks to find a binary vector $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y}) \in \{0,1\}^n \times \{0,1\}^k$ such that

$$\log_g(\boldsymbol{v}) \cdot \boldsymbol{z} = \log_g(\boldsymbol{a}) \cdot \boldsymbol{x} + \log_g(\boldsymbol{\ell}) \cdot \boldsymbol{y} = \log_g(X) \pmod{q-1},$$

where g is a generator of $\mathbb{Z}_q^\times$. One may then try to apply subset sum algorithms [19, 10] and hope that the first n coordinates of a solution correspond to the desired $\boldsymbol{x}$. However, this attack does not directly work for several reasons. First, even if a solution $\boldsymbol{z}$ to the subset sum instance is found, its first n entries need not be the correct solution of the original NSP, because the extra part $\boldsymbol{y}$ must encode exactly one noise prime and not an arbitrary subset. Second, the resulting density is not in the low-density range required by the classical attacks of [19, 10]. Third, even in low-density settings, those attacks rely on idealized short-basis subroutines stronger than what current lattice reduction algorithms can provably deliver. Hence this knapsack viewpoint does not currently yield an efficient algorithm against NSP.

Syndrome decoding. Following the third attack, now we use m NSP instances, where m is polynomial in n. This gives a multiple modular subset product instance $(\boldsymbol{V}, \boldsymbol{b})$, where each row is a subset product instance $(\boldsymbol{v}_i, X_i)$. This asks to find m different points of the form $\boldsymbol{z}_i = (\boldsymbol{x}, \boldsymbol{y}_i)$ for the m subset product instances respectively, with the constraints that each $\boldsymbol{y}_i$ has Hamming weight 1 and that the points all share the same $\boldsymbol{x}$. These are m points clustered within a small Hamming ball, with pairwise Hamming distance either 0 or 2. Again, no efficient reduction is currently known from this structured problem to a solved decoding task.

Beyond these failed attack attempts, we now show that for suitable L, Search-NSP is at least as hard as the LWE problem with uniformly random errors sampled from a generic error set.

Definition 4 (LWE). *Let S be a subset of $\mathbb{Z}_N$, and let $\boldsymbol{s} \leftarrow \{0,1\}^n$. Let $O_{\boldsymbol{s}}$ be the oracle that outputs pairs $(\boldsymbol{\alpha}, \beta)$ where $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_N^n$, $\varepsilon \leftarrow S$, and $\beta := \boldsymbol{\alpha} \cdot \boldsymbol{s} + \varepsilon \pmod{N}$. The LWE problem is, given $O_{\boldsymbol{s}}$, find $\boldsymbol{s}$.*

Theorem 2. *If half of the primes in L are quadratic residues modulo q, then LWE $\leq$ NSP.*

Proof. We reduce LWE with parameters (n, N, S) to Search-NSP with parameters (n, q, L), where $N = |\mathbb{Z}_q^\times| = q - 1$, $S = \log_g(L)$ is the set of discrete logarithms $\log_g(\ell_i)$ of the elements ℓ_i of L, and g is a generator of $\mathbb{Z}_q^\times$. By the assumption on L, the distribution D_L is exactly the uniform distribution over L.

We define a Search-NSP oracle $O_{\boldsymbol{x}}$ from the LWE oracle $O_{\boldsymbol{s}}$ as follows. For every LWE instance $(\boldsymbol{\alpha}, \beta)$ output by $O_{\boldsymbol{s}}$, the oracle $O_{\boldsymbol{x}}$ simply outputs $(\boldsymbol{a}, X) = \left(g^{\alpha_1} \bmod q, \ldots, g^{\alpha_n} \bmod q,\ g^{\beta} \bmod q\right)$. Note that

$$X = g^{\beta} = g^{\boldsymbol{\alpha} \cdot \boldsymbol{s} + \varepsilon} = \prod_{i=1}^{n} (g^{\alpha_i})^{s_i} \cdot g^{\varepsilon} = \boldsymbol{a}^{\boldsymbol{s}} \cdot e \pmod{q},$$

where $e = g^{\varepsilon} \bmod q$ is uniform over L, and hence is distributed according to D_L. It is clear that $O_{\boldsymbol{x}}$ is a Search-NSP oracle and that any solution $\boldsymbol{x}$ to the Search-NSP problem is a solution $\boldsymbol{s}$ to the LWE problem. In particular, if the LWE problem has a unique solution, then $\boldsymbol{x} = \boldsymbol{s}$. □

We now show that Decision-NSP is as hard as Search-NSP.

Lemma 1. *Search-NSP = Decision-NSP.*

Proof. This is the specialization to NSP of the corresponding result in the general MESS framework in [21, Lemma 4.6 and 4.7]. □

Based on the above discussion, we make the following assumption.

Assumption 1. *For every probabilistic polynomial time algorithm $\mathcal{A}$, there exists a negligible function $\mu(n)$ such that for all positive integers n, the probability that $\mathcal{A}$ solves Search-NSP is:*

$$\Pr[\mathcal{A}(O_{\boldsymbol{x}}) = \boldsymbol{x}] \leq \mu(n).$$

4 Bridging variants

Now we introduce several NSP variants and prove equivalence relations among them. These variants are tightly linked to the security proof of our key exchange scheme, as they bridge the hardness of Search-NSP and the hardness of the real problems that the scheme relies on.

4.1 Trapdoor NSP

The first bridging variants are the trapdoor variants Search-NSP$^\top$ and Decision-NSP$^\top$. They are obtained simply by replacing one uniformly random base vector in NSP with a structured noisy subset product vector coming from an independent NSP instance. By Assumption 1 and Lemma 1, such a structured vector is computationally indistinguishable from a uniformly random base vector.

Definition 5 (Trapdoor Instance). *An NSP instance is called a trapdoor instance if its base vector $\boldsymbol{v} \in (\mathbb{Z}_q^\times)^n$ is of the form $\boldsymbol{v} = S$ or $\boldsymbol{v} = S^{-1}$, where $S = (S_1, \ldots, S_n) = \boldsymbol{A}^{\boldsymbol{s}} \circ \boldsymbol{u} \pmod q$ is the vector of noisy products of an independent NSP instance with respect to an independent uniform secret $\boldsymbol{s} \leftarrow \{0,1\}^n$.*

Definition 6 (Search-NSP$^\top$). *Search-NSP with one of the instances a trapdoor instance.*

Definition 7 (Decision-NSP$^\top$). *Decision-NSP with one of the NSP instances a trapdoor instance.*

This trapdoor replacement does not change the hardness of the problem.

Lemma 2. *Decision-NSP $\leq$ Decision-NSP$^\top$.*

Proof. This is the specialization to NSP of the corresponding result in the general MESS framework [21]. □

Combining the preceding reductions gives the first equivalence that we need.

Theorem 3. *Search-NSP = Decision-NSP = Search-NSP$^\top$ = Decision-NSP$^\top$.*

Proof. By Lemma 1 we have Search-NSP = Decision-NSP. By Lemma 2 we have Decision-NSP $\leq$ Decision-NSP$^\top$. And obviously we have Decision-NSP$^\top$ $\leq$ Search-NSP$^\top$ and Search-NSP$^\top$ $\leq$ Search-NSP (by ignoring the trapdoor instance). Link them up to get the desired claim. □

4.2 Incomplete NSP

The second bridging variants are the incomplete variants Search-NSP$_\times$ and Decision-NSP$_\times$. They are simply Search-NSP and Decision-NSP "forgetting" one of the noises, namely one of the NSP instances is a noiseless subset product instance.

Definition 8 (Search-NSP$_\times$). *Search-NSP with one of the instances known to have noise $e = 1$.*

Definition 9 (Decision-NSP$_\times$). *Given $(O_{\boldsymbol{x}}, (\boldsymbol{a}, \boldsymbol{a}^{\boldsymbol{x}} \pmod q))$ or $(O_{ran}, (\boldsymbol{a}, r))$, where $\boldsymbol{a} \leftarrow (\mathbb{Z}_q^\times)^n$ and $r \leftarrow \mathbb{Z}_q^\times$, decide which is the case.*

Note that NSP$_\times$ can be viewed as NSP leaking one of the noises. When L is small, leaking a noise is expected. We therefore have the following lemma.

Lemma 3. *If $|L|$ is polynomial in n, then NSP = NSP$_\times$.*

Proof. The direction NSP$_\times \leq$ NSP is clear since we can always turn a subset product instance into a noisy subset product instance by multiplying a random noise. The other direction NSP $\leq$ NSP$_\times$ is also clear since if the size of L is polynomial in n then it is easy to guess one of the noises of the noisy subset product instances and turn the corresponding instance into a subset product instance by dividing out that noise. □

We also have the search-to-decision reduction for NSP$_\times$.

Lemma 4. *Search-NSP$_\times \leq$ Decision-NSP$_\times$.*

Proof. Similar to the proof of Lemma 1, where one of the noises being 1 does not affect the proof at all. □

4.3 Trapdoor incomplete NSP

The third bridging variants are combinations of the previous two variants, namely the trapdoor variants of Search-NSP$_\times$ and Decision-NSP$_\times$.

Definition 10 (Search-NSP$_\times^\top$). *Search-NSP$_\times$ with one of the instances a trapdoor instance.*

Definition 11 (Decision-NSP$_\times^\top$). *Decision-NSP$_\times$ with one of the NSP instances a trapdoor instance.*

Lemma 5. *Decision-NSP$_\times \leq$ Decision-NSP$_\times^\top$.*

Proof. Similar to the proof of Lemma 2, assuming the obvious fact that Decision-NSP$_\times \leq$ Decision-NSP (by ignoring the subset product instance). □

Hence we obtain the analogue of Theorem 3 for NSP$_\times$.

Theorem 4. *Search-NSP$_\times$ = Decision-NSP$_\times$ = Search-NSP$_\times^\top$ = Decision-NSP$_\times^\top$.*

Proof. By Lemma 4 we have Search-NSP$_\times$ = Decision-NSP$_\times$. By Lemma 5 we have Decision-NSP$_\times \leq$ Decision-NSP$_\times^\top$. And obviously we have Decision-NSP$_\times^\top \leq$ Search-NSP$_\times^\top$ and Search-NSP$_\times^\top \leq$ Search-NSP$_\times$ (by ignoring the trapdoor instance). □

4.4 Product computing

The fourth bridging variants are two problems analogous to the computational and decisional Diffie-Hellman problems (CDH/DDH). We call them the *computational/decisional product computing problems* (CPC/DPC). Their trapdoor variants are the real problems that our key exchange relies on.

Definition 12 (CPC). *Given $O_{\boldsymbol{x}}$ and $\boldsymbol{a} \leftarrow (\mathbb{Z}_q^\times)^n$, find $\boldsymbol{a}^{\boldsymbol{x}} \pmod q$.*

Definition 13 (DPC). *Given $(O_{\boldsymbol{x}}, (\boldsymbol{a}, \boldsymbol{a}^{\boldsymbol{x}} \pmod q))$ or $(O_{\boldsymbol{x}}, (\boldsymbol{a}, r))$, where $\boldsymbol{a} \leftarrow (\mathbb{Z}_q^\times)^n$ and $r \leftarrow \mathbb{Z}_q^\times$, decide which is the case.*

The difference between CPC and NSP is that NSP asks to find $\boldsymbol{x}$, while CPC only asks to find a subset product $\boldsymbol{a}^{\boldsymbol{x}} \pmod q$ with respect to $\boldsymbol{x}$. The difference between DPC and Decision-NSP$_\times$ is that Decision-NSP$_\times$ asks to distinguish many products from uniform, while DPC asks to distinguish only one product from uniform. We have the following lemma.

Lemma 6. *Decision-NSP$_\times \leq$ DPC.*

Proof. We prove it by a hybrid argument. Let D_0 and D_1 be the two distributions of Decision-NSP$_\times$, where D_0 is the NSP$_\times$ distribution and D_1 is the uniform distribution. Let D'_0 and D'_1 be the two distributions of DPC, where D'_0 is the CPC distribution and D'_1 is the uniform distribution. It is clear that $D_0 = D'_0$. Now notice that it is easy to convert D'_0 into D'_1 by replacing the subset product of D'_0 with a uniform element in $\mathbb{Z}_q^\times$, and it is easy to convert D'_1 into D_1 by replacing all the noisy products with uniform elements in $\mathbb{Z}_q^\times$. Hence D'_1 is an intermediate distribution between D'_0 and D_1. So if D_0 and D_1 are indistinguishable, that is, D'_0 and D_1 are indistinguishable, then by the triangle inequality, we must have that D'_0 and D'_1 are indistinguishable. □

Hence CPC and DPC are equivalent to NSP$_\times$.

Theorem 5. *Search-NSP$_\times$ = Decision-NSP$_\times$ = CPC = DPC.*

Proof. By Theorem 4 we have Search-NSP$_\times$ = Decision-NSP$_\times$. By Lemma 6 we have Decision-NSP$_\times$ $\leq$ DPC. And obviously we have DPC $\leq$ CPC and CPC $\leq$ Search-NSP$_\times$ (finding $\boldsymbol{x}$ immediately yields $\boldsymbol{a}^{\boldsymbol{x}} \pmod q$). □

4.5 Trapdoor product computing

Finally, we introduce the trapdoor variants of CPC and DPC. These are the final variants that are most directly used in the later security proofs.

Definition 14 (CPC$^\top$). *CPC with one of the instances a trapdoor NSP instance.*

Definition 15 (DPC$^\top$). *DPC with one of the NSP instances a trapdoor instance.*

Lemma 7. *Decision-NSP$_\times^\top \leq$ DPC$^\top$.*

Proof. Similar to the proof of Lemma 6, where the fact that there is an NSP instance being a trapdoor instance does not affect the proof at all. □

Theorem 6. *Search-NSP$_\times^\top$ = Decision-NSP$_\times^\top$ = DPC$^\top$ = CPC$^\top$.*

Proof. By Theorem 4 we have Search-NSP$_\times^\top$ = Decision-NSP$_\times^\top$. By Lemma 7 we have Decision-NSP$_\times^\top$ $\leq$ DPC$^\top$. And obviously we have DPC$^\top$ $\leq$ CPC$^\top$ and CPC$^\top$ $\leq$ Search-NSP$_\times^\top$. □

We also have the trapdoor relation on the product-computing side.

Lemma 8. *DPC $\leq$ DPC$^\top$.*

Proof. Similar to the proof of Lemma 2, assuming the obvious fact that DPC $\leq$ Decision-NSP (by ignoring the subset product instance). □

Combining the preceding equivalences yields the chain from the bridging variants to these final variants.

Theorem 7. *Search-NSP$_\times$ = Decision-NSP$_\times$ = Search-NSP$_\times^\top$ = Decision-NSP$_\times^\top$ = CPC = DPC = CPC$^\top$ = DPC$^\top$.*

Proof. By Theorem 5 we have Search-NSP$_\times$ = Decision-NSP$_\times$ = DPC. By Lemma 8 we have DPC $\leq$ DPC$^\top$. And obviously we have DPC$^\top$ $\leq$ CPC$^\top$ $\leq$ Search-NSP$_\times^\top$ $\leq$ Search-NSP$_\times$. The above problems form a reduction loop and thus are equivalent. Again, by Theorem 5 we have CPC = DPC and by Theorem 6 we have Decision-NSP$_\times^\top$ = DPC$^\top$. Hence all the problems involved are equivalent. □

4.6 Equivalence of all problems

When the noise set L is polynomially bounded, we obtain the full equivalence chain starting from Search-NSP.

Theorem 8. *If $|L|$ is polynomial in n, then: Search-NSP = Decision-NSP = Search-NSP$^\top$ = Decision-NSP$^\top$ = Search-NSP$_\times$ = Decision-NSP$_\times$ = Search-NSP$_\times^\top$ = Decision-NSP$_\times^\top$ = CPC = DPC = CPC$^\top$ = DPC$^\top$.*

Proof. If the size of L is polynomial in n, then Search-NSP = Search-NSP$_\times$ since: (1) Search-NSP$_\times \leq$ Search-NSP because we can always turn a Search-NSP$_\times$ oracle into a Search-NSP oracle by multiplying a random noise to the subset product or simply ignore the subset product instance; and (2) Search-NSP $\leq$ Search-NSP$_\times$ because if L is polynomial in n then we can always turn a Search-NSP oracle into a Search-NSP$_\times$ oracle by guessing one of the noises of the noisy subset product instances and turn the corresponding noisy subset product instance into a subset product instance by dividing out the noise. Having Search-NSP = Search-NSP$_\times$, we can then combine Theorem 3 and Theorem 7 to get our claim. □

4.7 Brief summary

For the reader's convenience, we summarize below the NSP-related problems introduced in this section, together with their intuitive meanings and their roles in the later security proof. By Theorem 8, when the noise set L is of polynomial size, all these formulations are equivalent to the basic Search-NSP problem.

Table 1. Summary of NSP variants and their roles.

Problem	Definition in words	Use later
Search-NSP	Recover $\boldsymbol{x}$ from noisy subset product instances.	Basic hardness assumption.
Decision-NSP	Distinguish NSP instances from uniform.	Search-to-decision bridge.
Search-NSP$^\top$	Search-NSP with one trapdoor instance.	Models trapdoor-shaped protocol components.
Decision-NSP$^\top$	Decision-NSP with one trapdoor instance.	Decisional trapdoor bridge.
Search-NSP$_\times$	Search-NSP with one noiseless instance.	Models shared-secret noiseless products such as $\boldsymbol{a}^{\boldsymbol{x}} \pmod q$.
Decision-NSP$_\times$	Decision version of Search-NSP$_\times$.	Intermediate decisional form.
Search-NSP$_\times^\top$	Search-NSP$_\times$ with one trapdoor instance.	Closest NSP-style abstraction of transcripts.
Decision-NSP$_\times^\top$	Decision version of Search-NSP$_\times^\top$.	Hybrid bridge to CPC/DPC.
CPC	Compute $\boldsymbol{a}^{\boldsymbol{x}} \pmod q$ from $O_{\boldsymbol{x}}$.	Shared secret recovery abstraction.
DPC	Distinguish $\boldsymbol{a}^{\boldsymbol{x}} \pmod q$ from random.	Shared secret indistinguishability abstraction.
CPC$^\top$	CPC with one trapdoor instance.	Used in Theorem 13.
DPC$^\top$	DPC with one trapdoor instance.	Used in the proof of standard security.

5 Key exchange

We are now well-prepared to present our *non-alternating key exchange* (NAKE) scheme, whose securities will be reduced to the hardness of Search-NSP using the results we proved in Section 3. More precisely, for

the security against shared secret recovery and the standard indistinguishability security, we work in the parameter regime where the noise set L is of polynomial size, so that the equivalent formulations collected in Theorem 8 apply.

We first introduce the syntax of NAKE and leave the formal security notions to be introduced later when they are needed for the security proofs. We define an NAKE scheme to be a pair of protocols $\Pi = (\Pi_0, \Pi_1)$, where Alice and Bob execute Π_0 to generate and publish public keys (execute once), and execute Π_1 to agree on keys non-alternatingly (can execute many times). Here, non-alternating means that Alice's and Bob's messages in each execution of Π_1 can be computed independently of each other and sent simultaneously. In other words, a NAKE is similar to Diffie-Hellman key exchange assuming Alice and Bob know each other's "flexible-term" public keys, whose lifetimes may be chosen in practice according to the desired tradeoff between non-alternatingness and forward secrecy.

5.1 The scheme

Let $pp = (\lambda, n, q, L, \boldsymbol{A}, \boldsymbol{a}, \boldsymbol{b})$ be public parameters, where $\boldsymbol{A} = \{a_{i,j}\}_{n\times n} \leftarrow D_a^{n\times n}$ and $\boldsymbol{a}, \boldsymbol{b} \leftarrow D_a^n$. The key idea of our scheme is that Alice and Bob respectively encode rows and columns of A into "row products" and "column products"; then the intersecting elements can be precisely cut off using their own secret keys, and the noises will be exposed. The scheme is shown by Fig. 6, where S, T, A, B are vectors and C, D, E, F are numbers, also sk and pk denote secret keys and public keys respectively.

Alice		**Bob**
KeyGen $\Pi_0(pp)$		
$sk_A : (\boldsymbol{s}, \boldsymbol{u}) \leftarrow \{0,1\}^n \times D_L^n$ $pk_A : S = \boldsymbol{A}^{\boldsymbol{s}} \circ \boldsymbol{u} \pmod q$		$sk_B : (\boldsymbol{t}, \boldsymbol{v}) \leftarrow \{0,1\}^n \times D_L^n$ $pk_B : T = {}^{\boldsymbol{t}}\boldsymbol{A} \circ \boldsymbol{v} \pmod q$
Key Exchange $\Pi_1(pp, S, T)$		
$(\boldsymbol{x}, \boldsymbol{e}, e) \leftarrow \{0,1\}^n \times D_L^n \times D_L$ $A := \boldsymbol{A}^{-\boldsymbol{x}} \circ \boldsymbol{e} \pmod q$ $C := T^{\boldsymbol{x}} \cdot e \pmod q$		$(\boldsymbol{y}, \boldsymbol{f}, f) \leftarrow \{0,1\}^n \times D_L^n \times D_L$ $B := {}^{-\boldsymbol{y}}\boldsymbol{A} \circ \boldsymbol{f} \pmod q$ $D := {}^{\boldsymbol{y}}S \cdot f \pmod q$
	$\xrightarrow{A, C}$	
	$\xleftarrow{B, D}$	
$E = B^{\boldsymbol{s}} \cdot D \pmod q$ $\boldsymbol{y}' = \boldsymbol{u}\vert E$ $K_A = (\boldsymbol{a}^{\boldsymbol{x}}, \boldsymbol{b}^{\boldsymbol{y}'}) \pmod q$		$F = {}^{\boldsymbol{t}}A \cdot C \pmod q$ $\boldsymbol{x}' = \boldsymbol{v}\vert F$ $K_B = (\boldsymbol{a}^{\boldsymbol{x}'}, \boldsymbol{b}^{\boldsymbol{y}}) \pmod q$

Fig. 6. Non-alternating key exchange from NSP.

Note that the inverse of Alice's message, i.e., $(A^{-1}, C^{-1}) \equiv (\boldsymbol{A}^{\boldsymbol{x}} \circ \boldsymbol{e}^{-1} \pmod q, (T^{-1})^{\boldsymbol{x}} \cdot e^{-1} \pmod q))$ is an NSP problem with base matrix $(\boldsymbol{A}^\top, (T^{-1})^\top)^\top$ and solution $\boldsymbol{x}$. Similarly, the inverse of Bob's message, i.e., $(B^{-1}, D^{-1}) \equiv ({}^{\boldsymbol{y}}\boldsymbol{A} \circ \boldsymbol{f}^{-1} \pmod q, {}^{\boldsymbol{y}}(S^{-1}) \cdot f^{-1} \pmod q)$ is an NSP problem with base matrix $(\boldsymbol{A}, (S^{-1})^\top)$ and solution $\boldsymbol{y}$. This observation will be used in the security proofs. A more detailed explanation of Fig. 6 is the following.

KeyGen $\Pi_0(pp)$:

$$\begin{matrix}\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} & u_1 \\ \vdots & & \vdots & \vdots \\ a_{n,1} & \dots & a_{n,n} & u_n \\ v_1 & \dots & v_n & 1 \end{bmatrix} & \begin{matrix} \xrightarrow{s} S_1 \\ \vdots \\ \xrightarrow{s} S_n \end{matrix} \\ \begin{matrix} \downarrow t & \cdots & \downarrow t \\ T_1 & \cdots & T_n \end{matrix} & \end{matrix}$$

- Alice: Sample secret key $(\boldsymbol{s}, \boldsymbol{u}) \leftarrow \{0,1\}^n \times D_L^n$. Compute an NSP product sequence $S = (S_1, \ldots, S_n)$, where $S_i = a_{i,1}^{s_1} \cdots a_{i,n}^{s_n} \cdot u_i \pmod q$ for $i \in \{1, \ldots, n\}$. Publish S as the public key.
- Bob: Sample secret key $(\boldsymbol{t}, \boldsymbol{v}) \leftarrow \{0,1\}^n \times D_L^n$. Compute an NSP product sequence $T = (T_1, \ldots, T_n)$, where $T_j = a_{1,j}^{t_1} \cdots a_{n,j}^{t_n} \cdot v_j \pmod q$ for $j \in \{1, \ldots, n\}$. Publish T as the public key.

Key Exchange $\Pi_1(pp, S, T)$:

$$\begin{matrix}\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} & e_1 \\ \vdots & & \vdots & \vdots \\ a_{n,1} & \dots & a_{n,n} & e_n \\ f_1 & \dots & f_n & 1 \end{bmatrix} & \begin{matrix} \xrightarrow{x} A_1 \\ \vdots \\ \xrightarrow{x} A_n \end{matrix} \\ \begin{matrix} \downarrow y & \cdots & \downarrow y \\ B_1 & \cdots & B_n \end{matrix} & \end{matrix}$$

- Alice: Sample ephemeral secret key $(\boldsymbol{x}, \boldsymbol{e}, e) \leftarrow \{0,1\}^n \times D_L^n \times D_L$. Compute an NSP product sequence $A = (A_1, \ldots, A_n)$, where $A_i := a_{i,1}^{-x_1} \cdots a_{i,n}^{-x_n} \cdot e_i \pmod q$ for $i \in \{1, \ldots, n\}$. Compute a composite NSP product $C := T_1^{x_1} \cdots T_n^{x_n} \cdot e \pmod q$. Send (A, C) to Bob.
- Bob (simultaneously): Sample ephemeral secret key $(\boldsymbol{y}, \boldsymbol{f}, f) \leftarrow \{0,1\}^n \times D_L^n \times D_L$. Compute an NSP product sequence $B = (B_1, \ldots, B_n)$, where $B_j := a_{1,j}^{-y_1} \cdots a_{n,j}^{-y_n} \cdot f_j \pmod q$, for $j \in \{1, \ldots, n\}$. Compute a composite NSP product $D := S_1^{y_1} \cdots S_n^{y_n} \cdot f \pmod q$. Send (B, D) to Alice.

Shared Secret Computation:

- Alice: Compute $E := B_1^{s_1} \cdots B_n^{s_n} \cdot D \pmod q$. Compute $\boldsymbol{y'} \in \{0,1\}^n$ such that $y'_j = 1$ if and only if $u_j | E$, for $j \in \{1, \ldots, n\}$. Set the shared secret to be $K_A = (\boldsymbol{a}^{\boldsymbol{x}} \pmod q, \boldsymbol{b}^{\boldsymbol{y'}} \pmod q)$.
- Bob: Compute $F := A_1^{t_1} \cdots A_n^{t_n} \cdot C \pmod q$. Compute $\boldsymbol{x'} \in \{0,1\}^n$ such that $x'_i = 1$ if and only if $v_i | F$, for $i \in \{1, \ldots, n\}$. Set the shared secret to be $K_B = (\boldsymbol{a}^{\boldsymbol{x'}} \pmod q, \boldsymbol{b}^{\boldsymbol{y}} \pmod q)$.

5.2 Correctness

Theorem 9. *If half of the primes in L are quadratic residues modulo q, then the key exchange succeeds (i.e. $K_A = K_B$) with probability greater than $1 - (4n+1)^2/k$, where $k = |L|$.*

Proof. By the setup in Section 3.1, the modulus q is chosen to be greater than ℓ_k^{2n+1}. Since every noise prime in L is at most ℓ_k, it follows that q is greater than the product of any $2n+1$ primes in L.

Therefore in the scheme we have

$$\begin{aligned} E &:= B^{\boldsymbol{s}} \cdot D \pmod q \\ &\equiv ({}^{\boldsymbol{y}}\boldsymbol{A} \circ \boldsymbol{f})^{\boldsymbol{s}} \cdot ({}^{\boldsymbol{y}}(\boldsymbol{A}^{\boldsymbol{s}}) \circ \boldsymbol{u}) \cdot f) \pmod q \\ &\equiv ({}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{s}})^{-1} \cdot \boldsymbol{f}^{\boldsymbol{s}} \cdot {}^{\boldsymbol{y}}\boldsymbol{A}^{\boldsymbol{s}} \cdot {}^{\boldsymbol{y}}\boldsymbol{u} \cdot f \pmod q \\ &\equiv \boldsymbol{f}^{\boldsymbol{s}} \cdot \boldsymbol{u}^{\boldsymbol{y}} \cdot f \pmod q \\ &\equiv (f_1^{s_1} \cdots f_n^{s_n}) \cdot (u_1^{y_1} \cdots u_n^{y_n}) \cdot f \pmod q \\ &= (f_1^{s_1} \cdots f_n^{s_n}) \cdot (u_1^{y_1} \cdots u_n^{y_n}) \cdot f, \end{aligned}$$

and

$$\begin{aligned} F &:= {}^{\boldsymbol{t}}A \cdot C \pmod q \\ &\equiv {}^{\boldsymbol{t}}(\boldsymbol{A}^{-\boldsymbol{x}} \circ \boldsymbol{e}) \cdot \left(({}^{\boldsymbol{t}}\boldsymbol{A} \circ \boldsymbol{v})^{\boldsymbol{x}} \cdot e\right) \pmod q \\ &\equiv ({}^{\boldsymbol{t}}\boldsymbol{A}^{\boldsymbol{x}})^{-1} \cdot {}^{\boldsymbol{t}}\boldsymbol{e} \cdot {}^{\boldsymbol{t}}\boldsymbol{A}^{\boldsymbol{x}} \cdot \boldsymbol{v}^{\boldsymbol{x}} \cdot e \pmod q \\ &\equiv \boldsymbol{e}^{\boldsymbol{t}} \cdot \boldsymbol{v}^{\boldsymbol{x}} \cdot e \pmod q \\ &\equiv \left(e_1^{t_1} \cdots e_n^{t_n}\right) \cdot \left(v_1^{x_1} \cdots v_n^{x_n}\right) \cdot e \pmod q \\ &= \left(e_1^{t_1} \cdots e_n^{t_n}\right) \cdot \left(v_1^{x_1} \cdots v_n^{x_n}\right) \cdot e, \end{aligned}$$

where in the last line of each calculation we used the fact that q is greater than the product of any $2n+1$ primes in L.

Now, if the noise primes u_i's, v_i's, e_i's, f_i's, e, and f are all different, then we have

$$y_j = 1 \iff u_j \mid E \qquad \text{and} \qquad x_i = 1 \iff v_i \mid F.$$

Hence $\boldsymbol{y}' = \boldsymbol{y}$ and $\boldsymbol{x}' = \boldsymbol{x}$, and therefore $K_A = K_B$. So the probability that $K_A = K_B$ is at least the probability that all these noise primes are different.

Now we sample only $4n+2$ noise primes from a set L of size k. Also, by the assumption on L, the distribution D_L is uniform over L. Hence, by the birthday-collision argument, the probability that they are all different is

$$\prod_{i=1}^{4n+1} \left(1 - \frac{i}{k}\right) > \left(1 - \frac{4n+1}{k}\right)^{4n+1} > 1 - \frac{(4n+1)^2}{k}.$$

Therefore the probability that $K_A = K_B$ is greater than $1 - (4n+1)^2/k$. □

In particular, the success probability is overwhelming if k is exponentially large. For example, subject to the condition on L in Theorem 9, if one takes $k = 2^n$, then $\Pr[K_A = K_B] > 1 - (4n+1)^2/2^n = 1 - 1/2^{n-2\log_2(4n+1)}$, which is overwhelming in n. In this exponential regime, the modulus q may also be estimated more concretely. By the prime number theorem, the m-th prime satisfies $\ell_m \sim m \log m$, and in particular one has $\ell_{2^n} \sim n2^n$. Hence, for $k = 2^n$, we have $q \gtrapprox (\ell_{2^n})^{2n+1} \sim (n2^n)^{2n+1}$. In fact, the asymptotic value $(n2^n)^{2n+1}$ is a rigorous upper bound for the product of any $2n+1$ primes in L. Indeed, an upper bound[6] [1] for the m-th prime ℓ_m for $m \geq 46\,254\,381$ is

$$\ell_m < m\left(\log m + \log\log m - 1 + \frac{\log\log m - 2}{\log m} - \frac{(\log\log m)^2 - 6\log\log m + 10.667}{2\log^2 m}\right).$$

The right-hand side is smaller than $m\log m + m\log\log m$. Plugging in $m = 2^n$, we get $\ell_{2^n} < 2^n \cdot (n\log 2 + \log n + \log\log 2) < n2^n$ for sufficiently large n. Hence all elements of L are smaller than $n2^n$, and therefore any prime q such that $q > (n2^n)^{2n+1}$ is guaranteed to be greater than the product of any $2n+1$ primes in L.

However, for the clean reduction of the shared-secret recovery and indistinguishability securities to the basic Search-NSP problem, we work in the parameter regime where $|L|$ is polynomial in n, so that Theorem 8 applies. In that regime, the lower bound in Theorem 9 is no longer negligible by itself.

Nonetheless, this does not mean that the practical correctness is poor. Even for moderately large polynomial choices of $|L|$, the decoding failure probability is already very small. For example, subject to the condition on L in Theorem 9, if $|L| = n^6$, then Theorem 9 gives failure probability at most $(4n+1)^2/n^6 = O(1/n^4)$. In particular, when $n = \lambda = 128$, we expect fewer than one decoding failure over a million key exchanges. When $n = \lambda = 1024$, we expect fewer than one decoding failure over a billion key exchanges.

[6] To the best of our knowledge, this is currently the tightest upper bound known.

5.3 Space and time complexity

Theorem 10. *The bit lengths of public keys and messages are both* $O\big(n^2 \log(k \log k)\big)$*, where* $k = |L|$*.*

Proof. By the setup in Section 3.1, we have $q > \ell_k^{2n+1}$. Hence $\log_2 q = O(n \log \ell_k)$. By the prime number theorem, the k-th prime satisfies $\ell_k = O(k \log k)$, and therefore $\log_2 q = O\big(n \log(k \log k)\big)$. Now Alice's public key S consists of n integers modulo q, and her message (A, C) consists of $n+1$ integers modulo q. Hence the bit lengths of her public key and message are both $O\big(n \log q\big) = O\big(n^2 \log(k \log k)\big)$. The situations for Bob's public key T and message (B, D) are the same. □

Note that the (seed of the) base matrix $\boldsymbol{A}$ and the public keys S and T do not need to be transmitted every time but once.

Theorem 11. *The computational complexity*[7] *of key generation and key exchange are both* $O\big(n^4 \log^2(k \log k)\big)$*, where* $k = |L|$*.*

Proof. The complexities mainly come from modular multiplications. By the proof above, we have $\log_2 q = O\big(n \log(k \log k)\big)$. Also notice that by the Montgomery method [22], the complexity of modular multiplication modulo a prime is upper bounded by the square of the bit length of the modulus. Hence the complexity of a single modular multiplication in our scheme is $O(\log_2^2 q) = O\big(n^2 \log^2(k \log k)\big)$. Now there are $O(n^2)$ modular multiplications in both key generation and key exchange. Hence the total time complexities of key generation and key exchange are both $O\big(n^4 \log^2(k \log k)\big)$. □

Therefore, even when k is exponential in n, and thus $q > \ell_k^{2n+1}$ is extremely large, the scheme still runs in polynomial time.

5.4 Security

We use a passive multi-session eavesdropping model with partial historical leakage. Concretely, the adversary is given all public transcripts, together with all historical ephemeral secrets except those of the challenge session. This captures a stronger setting than ordinary single-session passive security, while remaining weaker than active chosen-ciphertext style models.

Security notions. Let $\mathcal{A}$ be an adversary, λ be the security parameter, m be a positive integer polynomial in λ, sk_A, sk_B be secret keys, pk_A, pk_B be public keys, and pp be public parameters of the scheme which contains (but not restricted to) λ. Let $\mathcal{K}$ be the shared secret space of the NAKE scheme Π. Define the following experiment for NAKE:

Non-alternating key exchange experiment $\mathsf{NAKE}^{eva}_{\mathcal{A},\Pi}(pp, m)$:

- Two parties holding pp execute protocol Π_0 to generate key pairs (sk_A, pk_A) and (sk_B, pk_B) and exchange pk_A and pk_B. This results in a transcript trans_0 containing the public keys pk_A and pk_B.
- Two parties holding (pp, pk_A, pk_B) execute protocol Π_1 for m times. Each time results in a transcript $\mathsf{trans}_1^{(i)}$, $i \in \{1, \ldots, m\}$ containing all the messages sent by the parties in that execution of Π_1, and a shared secret K_i output by each of the parties. Denote the ephemeral secrets and randomness used in each execution of Π_1 by $\mathsf{eph}_1^{(i)}$, $i \in \{1, \ldots, m\}$.
- Choose m uniform bits $b_i \leftarrow \{0, 1\}$, $i \in \{1, \ldots, m\}$. If $b_i = 0$ set $\hat{K}_i := K_i$, and if $b_i = 1$ then choose $\hat{K}_i \leftarrow \mathcal{K}$ uniformly at random.
- Attacker $\mathcal{A}$ chooses an integer $k \in \{1, \ldots, m\}$, it is given pp, $\mathsf{trans} = (\mathsf{trans}_0, \mathsf{trans}_1^{(1)}, \ldots, \mathsf{trans}_1^{(\mathsf{m})})$, $K = (K_1, \ldots, K_{k-1}, \hat{K}_k, K_{k+1}, \ldots, K_m)$, $\mathsf{eph} = (\mathsf{eph}_1^{(1)}, \ldots, \mathsf{eph}_1^{(k-1)}, \mathsf{eph}_1^{(k+1)}, \ldots, \mathsf{eph}_1^{(m)})$ and it outputs a bit b'.

[7] Not including communication complexity.

- The output of the experiment is defined to be 1 if $b' = b_k$, and 0 otherwise. (In case $\mathsf{NAKE}^{eva}_{\mathcal{A},\Pi}(pp, m) = 1$, we say that $\mathcal{A}$ succeeds.)

In other words, the adversary $\mathcal{A}$ succeeds in breaking Π if it is given all historical transcripts trans and all historical secrets except for (sk_A, sk_B) and the secrets in the k-th execution of Π_1, and can correctly determine whether the key $\hat{K}_k$ is the real shared secret in the k-th execution of Π_1, or an independently uniform key, for any $k \in \{1, \ldots, m\}$ she chooses to work with. We say Π is secure if the adversary succeeds with probability that is at most negligibly greater than $1/2$. That is:

Definition 16 (Security of NAKE). *A NAKE Π is secure in the presence of an eavesdropper if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$ such that for all positive integers λ and for all positive integers m polynomial in λ, the probability that $\mathcal{A}$ succeeds is:*

$$\Pr[\mathsf{NAKE}^{eva}_{\mathcal{A},\Pi}(pp, m) = 1] \leq \frac{1}{2} + \mu(\lambda).$$

Note that this security notion is quite strong in the sense that it is stronger than the standard security of key exchange [18, Section 11.3, p. 391] as it considers multiple executions, and it also gives the adversary certain encoding and decoding abilities, similar to the IND-CPA [18, Section 12.3, p. 420] and IND-CCA [18, Section 12.3, p. 425] securities of KEM, with the difference that the adversary here can only decode encodings that are honestly generated from uniform secrets instead of arbitrary secrets.

We call the above security notion the *standard security* of NAKE. Following are two weaker notions about one-wayness.

Definition 17 (Security Against Secret Key Recovery). *A NAKE Π is secure against secret key recovery in the presence of an eavesdropper if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$ such that for all positive integers λ and for all positive integers m polynomial in λ, the probability that $\mathcal{A}$ finds a secret key is:*

$$\Pr[\mathcal{A}(pp, m, \mathsf{trans}) = sk_A \text{ or } sk_B] \leq \mu(\lambda).$$

Definition 18 (Security Against Shared Secret Recovery). *A NAKE Π is secure against shared secret recovery in the presence of an eavesdropper if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$ such that for all positive integers λ and for all positive integers m polynomial in λ, the probability that $\mathcal{A}$ finds a shared secret is:*

$$\Pr[\mathcal{A}(pp, m, \mathsf{trans}) \in \{K_1, \ldots, K_m\}] \leq \mu(\lambda).$$

Security proofs. Now we prove that our scheme achieves all the above three securities. The security against secret key recovery is based on the hardness of NSP. The security against shared secret recovery and the standard indistinguishability security are based on the equivalent formulations in Theorem 8, and hence reduce to the hardness of NSP when L is of polynomial size.

We first prove security against secret key recovery from Assumption 1.

Theorem 12. *If NSP is hard, then the NAKE $\Pi = (\Pi_0, \Pi_1)$ given by Fig. 6 is secure against secret key recovery.*

Proof. Since the scheme is symmetric, we only deal with sk_A. The case of sk_B is similar.

We want to show that for every probabilistic polynomial time algorithm $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$ such that for all positive integers λ and for all positive integers m polynomial in λ, the probability that $\mathcal{A}$ finds the secret key $sk_A = (\boldsymbol{s}, \boldsymbol{u})$ from $pp = (\lambda, n, q, L, \boldsymbol{A}, \boldsymbol{a}, \boldsymbol{b})$, m, and $\mathsf{trans} = (\mathsf{trans}_0 = (S, T), \mathsf{trans}_1 = (A^{(i)}, B^{(i)}, C^{(i)}, D^{(i)})_{i=1,\ldots,m})$ is: $p_1 := \Pr[\mathcal{A}(pp, m, \mathsf{trans}) = sk_A] \leq \mu(\lambda)$.

We will define an NSP algorithm $\mathcal{B}$ with success probability p_2 and show that there exists a negligible function $\mu(\lambda)$ such that $p_1 \leq p_2 \leq \mu(\lambda)$.

We define $\mathcal{B}$ as the following. It is given an NSP problem $(\boldsymbol{A}, S)$ with parameters n, q, L, where $\boldsymbol{A} \in (\mathbb{Z}_q^\times)^{n\times n}$, $S = (S_1, \ldots, S_n) \equiv \boldsymbol{A}^{\boldsymbol{s}} \circ \boldsymbol{u} \pmod q$, $\boldsymbol{s} \leftarrow \{0,1\}^n$, and $\boldsymbol{u} \leftarrow D_L^n$. It samples $\boldsymbol{a}, \boldsymbol{b} \leftarrow D_a^n$ and simulates the NAKE Π as follows. It treats S as the public key of Alice and computes the other transcripts T, $A^{(i)}$, $B^{(i)}$, $C^{(i)}$, $D^{(i)}$ of the scheme, where $i = 1, \ldots, m$ and m is the number of executions of Π_1. This can be done because $T, A^{(i)}, B^{(i)}, C^{(i)}$ only rely on $\boldsymbol{A}$, and $D^{(i)}$ only relies on S, but now both $\boldsymbol{A}$ and S are already given by the NSP instance. It then calls $\mathcal{A}$ with pp, m, and $\mathsf{trans} = (\mathsf{trans}_0 = (S, T), \mathsf{trans}_1 = (A^{(i)}, B^{(i)}, C^{(i)}, D^{(i)})_{i=1,\ldots,m})$ to find Alice's secret $(\boldsymbol{s}, \boldsymbol{u})$. It eventually outputs the first entry $\boldsymbol{s}'$ of $\mathcal{A}$'s output $(\boldsymbol{s}', \boldsymbol{u}')$ as its guess of the NSP solution $\boldsymbol{s}$.

Note that $\boldsymbol{s}' = \boldsymbol{s}$ if $(\boldsymbol{s}', \boldsymbol{u}')$ is the correct secret key $(\boldsymbol{s}, \boldsymbol{u})$ of Alice, which is true with probability p_1 by definition. Hence $p_1 \le p_2$. By the hardness of NSP (i.e. Assumption 1), there exists a negligible function $\nu(n)$ such that $p_2 \le \nu(n)$. Hence $p_1 \le \nu(n)$. Since n is polynomial in λ, we conclude that there exists a negligible function $\mu(\lambda)$ such that $p_1 \le \mu(\lambda)$. □

We then prove security against shared secret recovery from the hardness of CPC$^\top$, which is equivalent to NSP in the case where L is of polynomial size by Theorem 8.

Theorem 13. *If NSP is hard and $|L|$ is polynomial in n, then the NAKE $\Pi = (\Pi_0, \Pi_1)$ given by Fig. 6 is secure against shared secret recovery.*

Proof. We want to show that for every probabilistic polynomial time algorithm $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$ such that for all positive integers λ and for all positive integers m polynomial in λ, the probability that $\mathcal{A}$ finds any of the shared secrets $K_1, \ldots, K_m$ from $pp = (\lambda, n, q, L, \boldsymbol{A}, \boldsymbol{a}, \boldsymbol{b})$, m, and $\mathsf{trans} = (\mathsf{trans}_0 = (S, T), \mathsf{trans}_1 = (A^{(i)}, B^{(i)}, C^{(i)}, D^{(i)})_{i=1,\ldots,m})$ is: $p_1 := \Pr[\mathcal{A}(pp, m, \mathsf{trans}) \in \{K_1, \ldots, K_m\}] \le \mu(\lambda)$.

By Theorem 8, if NSP is hard and L is of polynomial size, then CPC$^\top$ is hard. Hence it is sufficient to reduce the desired security to the hardness of CPC$^\top$.

For this, we will define a CPC$^\top$ algorithm $\mathcal{B}$ with success probability p_2 and show that there exists a negligible function $\mu(\lambda)$ such that $p_1 \le p_2 \le \mu(\lambda)$.

We define $\mathcal{B}$ as the following. It is given a CPC$^\top$ problem

$$(\boldsymbol{C}, \boldsymbol{c}; \boldsymbol{a}) = ((\boldsymbol{A}^\top, (T^{-1})^\top)^\top, (A^{-1}, C^{-1}); \boldsymbol{a})$$

with parameters n, q, L, where $\boldsymbol{C} \in (\mathbb{Z}_q^\times)^{(n+1)\times n}$ is a base matrix with the first n rows the rows of $\boldsymbol{A} \in (\mathbb{Z}_q^\times)^{n\times n}$ and the last row $T^{-1} = (T_1^{-1}, \ldots, T_n^{-1}) \equiv ({}^{\boldsymbol{t}}\boldsymbol{A} \circ \boldsymbol{v} \pmod q)^{-1}$ a sequence of inverses of noisy products with respect to an independent vector $\boldsymbol{t} \leftarrow \{0,1\}^n$, and $\boldsymbol{c} = (A^{-1}, C^{-1}) = ((A_1^{-1}, \ldots, A_n^{-1}), C^{-1}) \equiv ((\boldsymbol{A}^{-\boldsymbol{x}} \circ \boldsymbol{e} \pmod q)^{-1}, ((T^{-1})^{-\boldsymbol{x}} \cdot e \pmod q)^{-1}) \equiv (\boldsymbol{A}^{\boldsymbol{x}} \circ \boldsymbol{e}^{-1} \pmod q, T^{\boldsymbol{x}} \cdot e^{-1} \pmod q)$ the sequence of noisy products with C^{-1} the trapdoor product whose base vector is T^{-1}.

Algorithm $\mathcal{B}$ simulates the NAKE scheme Π as follows. It samples $\boldsymbol{b} \leftarrow D_a^n$. Fixing the first session as the challenge session and losing a factor m, it chooses $k = 1$ and treats S and T as Alice and Bob's public keys, treats $(A, C) = (A^{(1)}, C^{(1)})$ and $(B, D) = (B^{(1)}, D^{(1)})$ as Alice's and Bob's messages in the first execution of Π_1, and computes the other transcripts $A^{(i)}$, $B^{(i)}$, $C^{(i)}$, $D^{(i)}$ of the scheme, where $i = 2, \ldots, m$ and m is the number of executions of Π_1. This can be done because only $C^{(i)}$ and $D^{(i)}$ rely on T and S, which are given by the CPC$^\top$ instance, and all the other transcripts are independent of $(S, T, A, B, C, D, \boldsymbol{a})$. It then calls $\mathcal{A}$ with pp, m, and $\mathsf{trans} = (\mathsf{trans}_0 = (S, T), \mathsf{trans}_1 = (A^{(i)}, B^{(i)}, C^{(i)}, D^{(i)})_{i=1,\ldots,m})$ to find a shared key $K_k = ((\boldsymbol{a}^{\boldsymbol{x}^{(k)}} \pmod q)', (\boldsymbol{b}^{\boldsymbol{y}^{(k)}} \pmod q)')$ for some $k \in \{1, \ldots, m\}$. It eventually outputs the first component $(\boldsymbol{a}^{\boldsymbol{x}^{(k)}} \pmod q)'$ as its guess of the CPC$^\top$ solution $\boldsymbol{a}^{\boldsymbol{x}^{(k)}} \pmod q$.

Note that $\mathcal{B}$ solves the CPC$^\top$ problem if $k = 1$ and the first component of K_1' is correct, which is true with probability at least p_1/m by definition of p_1. Hence $p_1/m \le p_2$. By the hardness of CPC$^\top$, there exists a negligible function $\nu(\lambda)$ such that $p_2 \le \nu(\lambda)$. Hence $p_1 \le m \cdot \nu(\lambda)$. Since m, n are polynomial in λ, we conclude that there exists a negligible function $\mu(\lambda)$ such that $p_1 \le \mu(\lambda)$. □

Now we prove the standard indistinguishable security of NAKE from the hardness of DPC$^\top$, which is equivalent to NSP in the case where L is of polynomial size by Theorem 8.

Theorem 14. *If NSP is hard and $|L|$ is polynomial in n, then the NAKE $\Pi = (\Pi_0, \Pi_1)$ given by Fig. 6 is secure.*

Proof. We want to show that for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$ such that for all positive integers λ and for all positive integers m polynomial in λ, the probability that $\mathcal{A}$ succeeds is: $\Pr[\mathsf{NAKE}^{eva}_{\mathcal{A},\Pi}(pp, m) = 1] \leq 1/2 + \mu(\lambda)$.

Define the advantage of $\mathcal{A}$ as $Adv(\mathcal{A}) := \Pr[\mathsf{NAKE}^{eva}_{\mathcal{A},\Pi}(pp, m)] - 1/2$, then what we want to show is $Adv(\mathcal{A}) \leq \mu(\lambda)$.

By Theorem 8, if NSP is hard and L is of polynomial size, then DPC$^\top$ is hard. Hence it is sufficient to reduce the desired security to the hardness of DPC$^\top$.

For this, we will define a DPC$^\top$ algorithm $\mathcal{B}$ which solves DPC$^\top$ with advantage $Adv(\mathcal{B}) := \Pr[\mathcal{B} \text{ solves DPC}^\top] - 1/2$ and show that there exists a negligible function $\mu(\lambda)$ such that $Adv(\mathcal{A}) \leq Adv(\mathcal{B}) \leq \mu(\lambda)$.

We define $\mathcal{B}$ as the following. It is given a DPC$^\top$ problem

$$(\boldsymbol{C}, \boldsymbol{c}; \boldsymbol{a}, r) = ((\boldsymbol{A}^\top, (T^{-1})^\top)^\top, (A^{-1}, C^{-1}); \boldsymbol{a}, r)$$

with parameters n, q, L, where $(\boldsymbol{C}, \boldsymbol{c})$ are as explained in the proof of Theorem 13, and r is either the subset product $r \equiv \boldsymbol{a}^{\boldsymbol{x}} \pmod q$, or a uniform number sampled from $\mathbb{Z}_q^\times$.

Algorithm $\mathcal{B}$ simulates the NAKE Π as follows. It samples $\boldsymbol{b} \leftarrow D_a^n$. Fixing the first session as the challenge session and losing a factor m, it chooses $k = 1$ and treats S and T as Alice and Bob's public keys, treats $(A, C) = (A^{(1)}, C^{(1)})$ and $(B, D) = (B^{(1)}, D^{(1)})$ as Alice's and Bob's messages in the first execution of Π_1, treats $(r, \boldsymbol{b}^{\boldsymbol{y}^{(1)}} \pmod q)$ as the challenge shared secret K_1, and computes the other transcripts $A^{(i)}$, $B^{(i)}$, $C^{(i)}$, $D^{(i)}$, K_i for $i = 2, \ldots, m$. This can be done because they are either independent of $(S, T, A, B, C, D, \boldsymbol{a}, r)$, or only rely on S or T, which are already given by the DPC$^\top$ instance. It then calls $\mathcal{A}$ with $pp = (\lambda, n, q, L, \boldsymbol{A}, \boldsymbol{a}, \boldsymbol{b})$, m, and $\mathsf{trans} = (\mathsf{trans}_0 = (S, T), \mathsf{trans}_1 = (A^{(i)}, B^{(i)}, C^{(i)}, D^{(i)}, K_i)_{i=1,\ldots,m})$, and outputs what $\mathcal{A}$ outputs.

Note that the DPC$^\top$ instance is a real shared secret instance if $K_1 := (r, \boldsymbol{b}^{\boldsymbol{y}^{(1)}} \pmod q)$ has first component $r = \boldsymbol{a}^{\boldsymbol{x}^{(1)}} \pmod q$, and it is a random instance if r is uniform. Hence $\mathcal{B}$ solves the given DPC$^\top$ if $\mathcal{A}$ chooses $k = 1$ and it succeeds in the NAKE experiment. Thus $Adv(\mathcal{A})/m \leq Adv(\mathcal{B})$. By the hardness of DPC$^\top$, there exists a negligible function $\nu(\lambda)$ such that $Adv(\mathcal{B}) \leq \nu(\lambda)$. Hence $Adv(\mathcal{A}) \leq m \cdot \nu(\lambda)$. Since m, n are polynomial in λ, we conclude that there exists a negligible function $\mu(\lambda)$ such that $Adv(\mathcal{A}) \leq \mu(\lambda)$. □

5.5 Remarks

We remark that a concrete advantage of recovering the sender's ephemeral secret key is that it naturally supports a multi-party non-alternating key exchange extension. Let $\boldsymbol{M} = (\boldsymbol{A}^\top, \boldsymbol{B}^\top)^\top$ be an $(n + N) \times n$ matrix whose last N rows are the N public keys of N parties. If Alice wants to share a secret with the other $N - 1$ parties, she may simply sample $\boldsymbol{x} \leftarrow \{0, 1\}^n$ and $\boldsymbol{a} \leftarrow (\mathbb{Z}_q^\times)^n$, and publish $(A := \boldsymbol{M}^{\boldsymbol{x}} \circ \boldsymbol{e} \pmod q, \boldsymbol{a})$. Then every party whose public key is embedded in $\boldsymbol{M}$ can recover the same $\boldsymbol{x}$ from $\boldsymbol{M}^{\boldsymbol{x}} \circ \boldsymbol{e} \pmod q$ and hence derive the same shared secret $K := \boldsymbol{a}^{\boldsymbol{x}} \pmod q$. In this sense, the recovered ephemeral secret key is not merely a hidden seed for one receiver, but a reusable shared resource for a whole group. By contrast, in a typical lattice-based viewpoint, one would usually encapsulate or encrypt the same auxiliary secret separately to different receivers. Here the same effect arises more directly from the recoverability of the ephemeral secret key itself.

We also emphasize that the main goal of this paper is to identify the underlying problem, develop the structural properties needed for cryptographic use, and show that it supports a post-quantum key exchange construction with clean security reductions. Accordingly, the present work is conceptual and reduction-theoretic rather than optimization-driven. A more refined study of concrete parameter optimization and practical implementation is left to future work.

6 Attacks against reused keys

We propose two attacks for our NAKE where Alice's ephemeral secret key $(\boldsymbol{x}, \boldsymbol{e})$ is reused in two different ways. The attacks against Bob are similar. This means that ephemeral secret keys should not be reused in our NAKE.

6.1 Reusing $\boldsymbol{x}$ with same $\boldsymbol{A}$

In this case, throughout different executions of the key exchange protocol, the secrets $\boldsymbol{x}$ are the same; the noise vectors $\boldsymbol{e}$ are different; the base matrices $\boldsymbol{A}$ are the same. The attack is based on the following theorem.

Theorem 15 (Diophantine Approximation [17]). *Let $\alpha \in \mathbb{R}$ then there exist fractions $a/b \in \mathbb{Q}$ such that $\left|\alpha - \frac{a}{b}\right| < \frac{1}{\sqrt{5}b^2}$. If, on the other hand, there exists $a/b \in \mathbb{Q}$ such that $\left|\alpha - \frac{a}{b}\right| < \frac{1}{2b^2}$, then a/b is a continued fraction convergent of α.*

The attack is as follows. Let $\boldsymbol{e}$ and $\boldsymbol{e}'$ be two ephemeral noise vectors of Alice in two executions of the protocol. To learn e_i and e'_i, take the two i-th NSP products $A_i := \boldsymbol{a}_i^{\boldsymbol{x}} \cdot e_i \pmod q$ and $A'_i := \boldsymbol{a}_i^{\boldsymbol{x}} \cdot e'_i \pmod q$ in the two executions, where $\boldsymbol{a}_i$ is the i-th row of $\boldsymbol{A}$. Compute

$$E := A \cdot A'^{-1} \equiv e_i \cdot e_i'^{-1} \pmod q.$$

We have that $Ee'_i - kq = e_i$ hence $\left|\frac{E}{q} - \frac{k}{e'_i}\right| = \frac{e_i}{qe'_i}$. Since q is greater than the product of any $2n+1$ noise primes, we have that $q > 2e_ie'_i$. I.e., $\frac{e_i}{qe'_i} < \frac{1}{2e_i'^2}$. By Theorem 15, $\frac{k}{e'_i}$ is a convergent of $\frac{E}{q}$. We can then find the convergent $\frac{k}{e'_i}$ from the continued fraction of $\frac{E}{q}$. This search is efficient since the depth of the continued fraction of $\frac{E}{q}$ is $\leq \log_2 q$. Then we obtain e'_i and k, as well as $e_i = Ee'_i - kq$.

6.2 Reusing $(\boldsymbol{x}, \boldsymbol{e})$ with different $\boldsymbol{A}$

In this case, throughout different executions of the protocol, both $\boldsymbol{x}$ and $\boldsymbol{e}$ are the same, but the base matrices $\boldsymbol{A}$ are different and random.

The attack is as follows. From two executions of the protocol, we have two messages from Alice, they are $A := \boldsymbol{A}^{\boldsymbol{x}} \circ \boldsymbol{e} \pmod q$ and $A' := \boldsymbol{A'}^{\boldsymbol{x}} \circ \boldsymbol{e} \pmod q$. Compute

$$\boldsymbol{c} := AA'^{-1} \equiv (\boldsymbol{A}\boldsymbol{A'}^{-1})^{\boldsymbol{x}} := \boldsymbol{B}^{\boldsymbol{x}} \pmod q$$

Now we borrow a DLP algorithm to take discrete logarithms entry-wise for $\boldsymbol{B}$ and $\boldsymbol{c}$ and we have

$$(\log_g \boldsymbol{B}) \cdot \boldsymbol{x} = \log_g \boldsymbol{c} \pmod{q-1},$$

where g is a generator of $\mathbb{Z}_q^\times$. Denote

$$\boldsymbol{H} := \log_g(\boldsymbol{B}) \text{ and } \boldsymbol{h} := \log_g \boldsymbol{c},$$

we have

$$\boldsymbol{H}\boldsymbol{x} = \boldsymbol{h} \pmod{q-1}.$$

Let p be the greatest prime factor of $q-1$ (if q is a safe prime then $p = (q-1)/2$), we have

$$\boldsymbol{H}\boldsymbol{x} = \boldsymbol{h} \pmod p.$$

By [6, 9] the probability that a uniform matrix in $\mathbb{Z}_p^{k\times n}$ with $k \geq n$ is of full $\mathbb{Z}_p$-rank n is

$$\prod_{i=k-n+1}^{k} \left(1 - \frac{1}{p^i}\right).$$

In our case where $k = n$, this probability is

$$\prod_{i=1}^{n}\left(1-\frac{1}{p^i}\right) > \left(1-\frac{1}{p}\right)^n > 1-\frac{n}{p}.$$

If q is a safe prime then $p = (q-1)/2$. Since by the setup we have $q > \ell_k^{2n+1}$, it follows that p is exponentially large in the relevant parameter range, and therefore the above probability is overwhelming in n. Hence with overwhelming probability we can solve the matrix equation $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{h} \pmod p$ for the unique $\boldsymbol{x}$ by Gaussian elimination efficiently.

Note that in this attack we take discrete logarithms rather than using Legendre symbols for turning exponential equations into linear equations. This is for the sake of having a high rank $\boldsymbol{H}$. Otherwise if we use Legendre symbols, the matrix $\boldsymbol{H}$ will be over $\mathbb{Z}_2$, which is of full $\mathbb{Z}_2$-rank with probability $\prod_{i=1}^{n}\left(1-\frac{1}{2^i}\right)$, which is only ≈ 0.288788 as $n \to \infty$, not a very promising probability. However, if q is a Mersenne prime then the greatest prime factor of $q-1$ is $p = 2$ and thus the full-rank probabilities will be the same. In that case, it is better to take Legendre symbols and save the use of a (quantum) DLP algorithm.

Acknowledgements

The author thanks Steven Galbraith and Libin Wang for valuable discussions.

References

[1] Christian Axler. "New Estimates for the nth Prime Number". In: *Journal of Integer Sequences* 22.2 (2019), p. 3.

[2] James Bartusek, Tancrède Lepoint, Fermi Ma, and Mark Zhandry. "New Techniques for Obfuscating Conjunctions". In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Yuval Ishai and Vincent Rijmen. Cham: Springer International Publishing, 2019, pp. 636–666. ISBN: 978-3-030-17659-4.

[3] E.R. Berlekamp. "The technology of error-correcting codes". In: *Proceedings of the IEEE* 68.5 (1980), pp. 564–593. DOI: 10.1109/PROC.1980.11696.

[4] Allison Bishop, Lucas Kowalczyk, Tal Malkin, Valerio Pastro, Mariana Raykova, and Kevin Shi. "A simple obfuscation scheme for pattern-matching with wildcards". In: *Annual International Cryptology Conference (CRYPTO)*. Springer. 2018, pp. 731–752.

[5] Ian F Blake and Chris Studholme. "Properties of random matrices and applications". In: *Unpublished report available at http://www. cs. toronto. edu/˜ cvs/coding* (2006).

[6] Johannes Blömer, Richard Karp, and Emo Welzl. "The rank of sparse random matrices over finite fields". In: *Random Structures & Algorithms* 10.4 (1997), pp. 407–419.

[7] Avrim Blum, Adam Kalai, and Hal Wasserman. "Noise-tolerant learning, the parity problem, and the statistical query model". In: *Journal of the ACM (JACM)* 50.4 (2003), pp. 506–519.

[8] Ran Canetti, Benjamin Fuller, Omer Paneth, Leonid Reyzin, and Adam Smith. "Reusable fuzzy extractors for low-entropy distributions". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2016, pp. 117–146.

[9] Colin Cooper. "On the distribution of rank of a random matrix over a finite field". In: *Random Structures & Algorithms* 17.3-4 (2000), pp. 197–212.

[10] Matthijs J Coster, Antoine Joux, Brian A LaMacchia, Andrew M Odlyzko, Claus-Peter Schnorr, and Jacques Stern. "Improved low-density subset sum algorithms". In: *Computational complexity* 2.2 (1992), pp. 111–128.

[11] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data". In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by Christian Cachin and Jan L. Camenisch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 523–540. ISBN: 978-3-540-24676-3.

[12] Andrej Dujella. "A variant of Wiener's attack on RSA". In: *Computing* 85.1-2 (2009), pp. 77–83.
[13] Andrej Dujella. "Continued fractions and RSA with small secret exponent". In: *arXiv preprint cs/0402052* (2004).
[14] Steven D. Galbraith and Trey Li. *Small Superset and Big Subset Obfuscation*. Cryptology ePrint Archive, Report 2020/1018. https://ia.cr/2020/1018. 2020.
[15] Steven D. Galbraith and Lukas Zobernig. "Obfuscated Fuzzy Hamming Distance and Conjunctions from Subset Product Problem". submitted.
[16] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 174. freeman San Francisco, 1979.
[17] Adolf Hurwitz. "Über die angenäherte Darstellung der Irrationalzahlen durch rationale Brüche". In: *Mathematische Annalen* 39.2 (1891), pp. 279–284.
[18] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. 3rd. Chapman & Hall/CRC, 2020.
[19] Jeffrey C Lagarias and Andrew M Odlyzko. "Solving low-density subset sum problems". In: *Journal of the ACM (JACM)* 32.1 (1985), pp. 229–246.
[20] Georg Landsberg. "Ueber eine Anzahlbestimmung und eine damit zusammenhängende Reihe." In: *Journal für die reine und angewandte Mathematik* 111 (1893), pp. 87–88. URL: http://eudml.org/doc/148874.
[21] Trey Li. "Multivariate exponential equations with unknown coefficients". In: *Advances in Mathematics of Communications* (2025), pp. 0–0.
[22] Peter L Montgomery. "Modular multiplication without trial division". In: *Mathematics of computation* 44.170 (1985), pp. 519–521.

A More applications of NSP

We discuss more applications of NSP to show its strong applicability. In particular, the applications about secure sketch, (reusable) fuzzy extractor and obfuscation cannot be achieved in such an easy way using *additive* noisy learning problems such as LPN or LWE.

As a concise preview, the public key encryption (PKE) scheme uses NSP with the noise vector $\boldsymbol{e}$ a random permutation of n or $n+1$ primes; the signature scheme uses NSP with freer (smaller) modulus q; the reusable fuzzy extractor and obfuscations use NSP with prime number bases.

A.1 PKE from NSP

A PKE can be extracted from our key exchange scheme if we take a one-sided look at the scheme.[8] The encryption idea is from the observation that the hardness of NSP does not seem to be affected if we use different forms of noise terms, as long as their entropy is high. Specifically, the noise terms can be (1) primes or composites; (2) sampled from one set or another; (3) with exponents $+1$ or -1; etc. We can therefore store information in the noise terms and that after recovering the noise terms, we can recover the i-th bit of the plaintext by checking if the i-th noise term is: (1) a prime or a composite; (2) a prime in the first noise set or the second; or (3) a prime in the noise set or the inverse of a prime in the noise set.

We take case (3) as an example to illustrate the idea. We first construct two trapdoor functions using NSP with free noise exponents of ± 1 and then turn them into two PKEs.

We also remark that one cannot encrypt the message by using it as the recoverable secret $\boldsymbol{x}$ of an NSP problem, since the hardness of NSP requires $\boldsymbol{x}$ to be drawn uniformly at random, whereas the message distribution need not satisfy this requirement.

[8] Note that our key exchange encoding-decoding is not immediately a PKE because the encoding algorithm only encodes a uniform secret $\boldsymbol{x} \in \{0,1\}^n$ (we do not claim hardness of NSP for all secret distributions), and so we cannot treat a plaintext $\boldsymbol{m} \in \{0,1\}^n$ (which is usually not uniform) as an NSP secret and encrypt it into NSP instances just like encoding $\boldsymbol{x}$.

Trapdoor functions from NSP Denote $\{0,1\}^n_\eta$ as the set of binary vectors with Hamming weight upper bounded by $\eta := n/(4\log_2 n)$; this is the domain of the secret $\boldsymbol{s}$. Besides, we need to use two sets of primes for two noise domains, we denote them as two vectors, they are the vector $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_n)$ of the first n primes, and the vector $\boldsymbol{t} = (t_1, \ldots, t_{n+1})$ of the next $n+1$ primes. One last important thing we need is a prime modulus q that is greater than the product of any $2\eta+1$ primes in $\boldsymbol{\ell}$ and $\boldsymbol{t}$. Now let $\boldsymbol{r}, \boldsymbol{s} \in \{0,1\}^n_\eta$ be two secrets; and $\boldsymbol{u}, \boldsymbol{e} := (\boldsymbol{v}, v)$ be two noise vectors which are random permutations of $\boldsymbol{\ell}$ and $\boldsymbol{t}$, respectively. Let $\boldsymbol{a} = \boldsymbol{A}^{\boldsymbol{r}} \circ \boldsymbol{u} := (\boldsymbol{a}_1^{\boldsymbol{r}} \cdot u_1, \ldots, \boldsymbol{a}_n^{\boldsymbol{r}} \cdot u_n)^\top \in (\mathbb{Z}_q^\times)^n$, where $\boldsymbol{a}_i$ are the rows of $\boldsymbol{A}$. Append $\boldsymbol{a}$ to the bottom of $\boldsymbol{A}$ and denote $\bar{\boldsymbol{A}} := (\boldsymbol{A}^\top, \boldsymbol{a}^\top)^\top$. The first NSP trapdoor function, called NSP* function, is the following:

$$f_{\bar{\boldsymbol{A}}}(\boldsymbol{s}, \boldsymbol{e}) = ({}^{\boldsymbol{s}}\boldsymbol{A} \circ \boldsymbol{v}^{-1} \pmod q, {}^{\boldsymbol{s}}\boldsymbol{a} \cdot v \pmod q),$$

where $\boldsymbol{v}^{-1} = (v_1^{-1}, \ldots, v_n^{-1})$, ${}^{\boldsymbol{s}}\boldsymbol{a} = \boldsymbol{a}^{\boldsymbol{s}} = \prod_{i=1}^n a_i^{s_i}$, ${}^{\boldsymbol{s}}\boldsymbol{A} = (\boldsymbol{c}_1^{\boldsymbol{s}}, \ldots, \boldsymbol{c}_n^{\boldsymbol{s}})$, and $\boldsymbol{c}_i$ are the columns of $\boldsymbol{A}$. Also, the trapdoor is $(\boldsymbol{r}, \boldsymbol{u})$. To recover $\boldsymbol{s}$ with $(\boldsymbol{r}, \boldsymbol{u})$, simply compute $E = ({}^{\boldsymbol{s}}\boldsymbol{A} \circ \boldsymbol{v}^{-1})^{-\boldsymbol{r}} \cdot ({}^{\boldsymbol{s}}\boldsymbol{a} \cdot v) \pmod q = u_1^{s_1} \cdots u_n^{s_n} \cdot v_1^{r_1} \cdots v_n^{r_n} \cdot v \pmod q$, which is a product of less than $2\eta+1$ primes in $\boldsymbol{\ell}$ and $\boldsymbol{t}$ modulo q. Recall that q is greater than the product of any $2\eta+1$ primes in $\boldsymbol{\ell}$ and $\boldsymbol{t}$, we can therefore ignore "mod q" and we have that $E = u_1^{s_1} \cdots u_n^{s_n} \cdot v_1^{r_1} \cdots v_n^{r_n} \cdot v$, and then we can recover $\boldsymbol{s}$ by testing which primes u_i in $\boldsymbol{u}$ divide E. It is not hard to see that $s_i = 1$ if and only if $u_i | E$.

The second NSP trapdoor function, called the NSP** function, allows the exponents of the v_i's and v to be ± 1 freely. It is defined as the following:

$$f_{\bar{\boldsymbol{A}}}(\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{\epsilon}) = ({}^{\boldsymbol{s}}\boldsymbol{A} \circ (\boldsymbol{v}^{\circ \boldsymbol{w}})^{-1} \pmod q, {}^{\boldsymbol{s}}\boldsymbol{a} \cdot v^w \pmod q),$$

where $\boldsymbol{\epsilon} = (\boldsymbol{w}, w) \in \{\pm 1\}^{n+1}$, and $\circ$ means entry-wise multiplication or entry-wise exponentiation. The main difference between NSP* and NSP** is that in NSP**, the recovery of $\boldsymbol{s}$ requires to factor $E = u_1^{s_1} \cdots u_n^{s_n} \cdot v_1^{\pm r_1} \cdots v_n^{\pm r_n} \cdot v^{\pm 1} \pmod q$, which (possibly) contains negative exponents. This can be handled by the following extended Legendre theorem:

Theorem 16 (Extended Legendre Theorem [13, 12]). *. Let α be an irrational number, let the fractions $\frac{p_i}{q_i} \in \mathbb{Q}$ be its continued fraction, and let a, b be coprime nonzero integers satisfying the inequality $\left|\alpha - \frac{a}{b}\right| < \frac{a}{b^2}$, where c is a positive real number. Then $(a, b) = (sp_{j+1} \pm tp_j, sq_{j+1} \pm tq_j)$, for some nonnegative integers j, r and s such that $st < 2c$.*

Theorem 16 is and extension of Theorem 15, it simply says that if E is of the form $U/V \pmod q$ for a (reduced) rational number U/V (i.e. $gcd(U, V) = 1$), we can still factor U and V if q is slightly larger than before, namely a constant times the product UV of the nominator U and denominator V of the rational U/V.

PKE from NSP Now we turn the two trapdoor functions into two PKEs. For the first PKE, we simply encode each bit $m_i \in \{0,1\}$ of a plaintext $\boldsymbol{m} \in \{0,1\}^k$ by sampling a uniform vector $\hat{\boldsymbol{r}} \in \{0,1\}^n$ and a random secret $\boldsymbol{s} \in \{0,1\}^n_\eta$ such that $\boldsymbol{s} \cdot \hat{\boldsymbol{r}} = m_i \pmod 2$ (this is expected to success in two samplings of $\boldsymbol{s}$ and does not greatly affect the randomness of $\boldsymbol{s}$). Then by a simple extension of the Goldreich-Leven theorem, the bit m_i is a hardcore bit if $\boldsymbol{e}$ is easy to recover from $f_{\bar{\boldsymbol{A}}(\boldsymbol{s}, \boldsymbol{e})}$ and $\boldsymbol{s}$, which is clearly true. The receiver can then recover m_i as $m_i = \boldsymbol{s} \cdot \bar{\boldsymbol{r}} \pmod 2$ after recovering the secret $\boldsymbol{s}$ of the sender using the trapdoor $(\boldsymbol{r}, \boldsymbol{u})$.

For the second PKE, we store an $n+1$ bit plaintext $\boldsymbol{m} \in \{0,1\}^{n+1}$ in the exponent vector $\boldsymbol{\epsilon}$ of $\boldsymbol{e}$ as $\boldsymbol{e}^{\boldsymbol{\epsilon}} = \boldsymbol{e}^{\circ(2\boldsymbol{m}-1)} := (v_1^{2m_1-1}, \ldots, v_n^{2m_n-1}, v^{2m_{n+1}-1})$, then after recovering $\boldsymbol{e}^{\circ(2\boldsymbol{m}-1)}$ using the trapdoor $(\boldsymbol{r}, \boldsymbol{u})$, the receiver can recover each bit of $\boldsymbol{m}$ by checking if $v_i^{2m_i-1}$ (resp. $v^{2m_{n+1}-1}$) is some prime t_j in $\boldsymbol{t}$, or the reciprocal t_j^{-1} of some prime t_j in $\boldsymbol{t}$.

A.2 Digital signature from NSP

Now we show how to construct a signature scheme using NSP. The strategy is to firstly construct an analogy of the Schnorr identification scheme by replacing DLP by NSP, then use the Fiat-Shamir transformation to make it a signature scheme.

Identification scheme from NSP Recall that an identification scheme involves two parties, the prover and the verifier. Both of them have access to the same public key. The prover wants to convince the verifier that she owns the corresponding private key of the public key. They interact as the following. The prover first sends a commitment to the verifier; the verifier then sends a challenge to the prover; the prover then sends a response to the verifier; in the end the verifier accepts or rejects the prover's claim about her possession of the secret key. The identification scheme from NSP is the following.

Let $m, n \in \mathbb{N}$ be positive integers with $m \geq 2n$, and q be a prime. Note that here we do not require q to be very large, because unlike our key exchange scheme, there is no need to decode anything here. In other words, the q here can be much smaller than the q in our key exchange scheme. Let $\boldsymbol{A} \in (\mathbb{Z}_q^\times)^{m \times n}$ be a uniform base matrix. The prover's private key is $(\boldsymbol{s}, \boldsymbol{u}) \leftarrow \mathbb{Z}_{q-1}^n \times L^m$. Her public key is $(\boldsymbol{A}, S := \boldsymbol{A}^{\boldsymbol{s}} \circ \boldsymbol{u} \pmod q)$.

- The prover samples $(\boldsymbol{x}, \boldsymbol{e}) \leftarrow \mathbb{Z}_{q-1}^n \times L^m$; computes $A = \boldsymbol{A}^{\boldsymbol{x}} \circ \boldsymbol{e} \pmod q$; and sends A to the verifier as the commitment;
- The verifier samples $c \leftarrow \mathbb{Z}_{q-1}$ and send it to the prover as the challenge;
- The prover computes $\boldsymbol{y} = \boldsymbol{x} - c\boldsymbol{s} \pmod{q-1}$ and $\boldsymbol{v} = \boldsymbol{e} \circ \boldsymbol{u}^{-c} \pmod q$, and sends $(\boldsymbol{y}, \boldsymbol{v})$ to the verifier as the response;
- The verifier computes $A' = \boldsymbol{A}^{\boldsymbol{y}} \circ S^c \circ \boldsymbol{v} \pmod q$; and accepts if $A' = A$ or rejects if $A' \neq A$.

It is clear that if every party in the identification scheme is honest then $A' = A$ because $A' = \boldsymbol{A}^{\boldsymbol{y}} \circ S_i^c \circ \boldsymbol{v} \mod q = \boldsymbol{A}^{\boldsymbol{y}} \circ (\boldsymbol{A}^{\boldsymbol{s}} \circ \boldsymbol{u})^c \circ (\boldsymbol{e} \circ \boldsymbol{u}^{-c}) \mod q = \boldsymbol{A}^{\boldsymbol{y}+c\boldsymbol{s}} \circ \boldsymbol{u}^c \circ (\boldsymbol{e} \circ \boldsymbol{u}^{-c}) \mod q = \boldsymbol{A}^{\boldsymbol{x}} \circ \boldsymbol{e} \mod q = A$. The security reduces to the hardness of NSP by a similar proof logic as in [18, Section 13.5.2], with a twist of the underlying computational problem. Here we do not go into details.

Signature scheme from NSP With the Fiat-Shamir transformation, the signature scheme is then the following, where the parameters n, m, q are the same as the identification scheme.

$\mathsf{KeyGen}(n, m, q)$:

- Sample $\boldsymbol{A} \leftarrow (\mathbb{Z}_q^\times)^{m \times n}$;
- Sample $(\boldsymbol{s}, \boldsymbol{u}) \leftarrow \mathbb{Z}_{q-1}^n \times L^m$;
- Compute $S = \boldsymbol{A}^{\boldsymbol{s}} \circ \boldsymbol{u} \pmod q$;
- Output (sk, pk) with $sk := (\boldsymbol{s}, \boldsymbol{u})$, $pk := (\boldsymbol{A}, S)$.

$\mathsf{Sign}(sk, a)$:

- Sample $(\boldsymbol{x}, \boldsymbol{e}) \leftarrow \mathbb{Z}_{q-1}^n \times L^m$ and compute $A = \boldsymbol{A}^{\boldsymbol{x}} \circ \boldsymbol{e} \pmod q$;
- Compute $c = H(A, a)$, where H is a cryptographic hash function;
- Compute $\boldsymbol{y} = \boldsymbol{x} - c\boldsymbol{s} \pmod{q-1}$ and $v = \boldsymbol{e} \circ \boldsymbol{u}^{-c} \pmod q$;
- Output $(\boldsymbol{y}, \boldsymbol{v}, c)$ as the signature.

$\mathsf{Verify}(pk, a, \boldsymbol{y}, \boldsymbol{v}, c)$:

- Compute $A' = \boldsymbol{A}^{\boldsymbol{y}} \circ S^c \circ \boldsymbol{v} \pmod q$;
- Compute $c' = H(A', a)$;
- Accept if $c' = c$ or reject if $c' \neq c$.

It is clear that if every party in the signature scheme is honest then $c' = c$ because $A' = A$ as we have seen in the identification scheme. The security follows from the security of the identification scheme and the Fiat-Shamir transformation.

A.3 Reusable fuzzy extractor from NSP

Secure sketches and fuzzy extractors were firstly introduced by Dodis, Reyzin and Smith in [11]. The idea of secure sketch is to encode a metric point $\boldsymbol{x}$ such that only a point $\boldsymbol{y}$ that is close to $\boldsymbol{x}$ can recover $\boldsymbol{x}$. The idea of fuzzy extractor is to encode a pseudo-random string $\boldsymbol{s}$ using a metric point $\boldsymbol{x}$ such that only a point $\boldsymbol{y}$ close to $\boldsymbol{x}$ can recover s with the help of some helper information $\boldsymbol{h}$. Reusable fuzzy extractor [8] further requires that the secret $\boldsymbol{x}$ is protected even given many encodings of it.

Note that there is a generic construction of fuzzy extractors from secure sketches [11]. The idea is to creat $\boldsymbol{s}$ by $\boldsymbol{x}$ together with a pseudo-random string $\boldsymbol{h}$ (e.g. take XOR as: $\boldsymbol{s} = \boldsymbol{x} \oplus \boldsymbol{h}$), sketch $\boldsymbol{x}$ as $\mathsf{SK}(\boldsymbol{x})$, and publish $\mathsf{SK}(\boldsymbol{x})$ and $\boldsymbol{h}$. Then with a point $\boldsymbol{y}$ close to $\boldsymbol{x}$ one could recover $\boldsymbol{x}$ from $\mathsf{SK}(\boldsymbol{x})$ and then recover $\boldsymbol{s} = \boldsymbol{x} \oplus \boldsymbol{h}$. Therefore in the following we only discuss reusable secure sketches, and that reusable fuzzy extractors are implied.

For a real life example, one can think of secure sketch as a fingerprint identification system, where the user register her fingerprint $\boldsymbol{x}$ in here cellphone the first time when she uses the phone. Then to unlock the phone, she enters her fingerprint $\boldsymbol{y}$. If $\boldsymbol{y} \approx \boldsymbol{x}$ then the phone unlocks; otherwise the phone remains locked.

An easy way to achieve this fingerprint identification system is to use NSP to encode the fingerprint $\boldsymbol{x}$ as an NSP instance $(\boldsymbol{a}, \mathsf{SK}(\boldsymbol{x}) = \boldsymbol{a}^{\boldsymbol{x}} \cdot e \pmod q := X)$, where the base vector $\boldsymbol{a}$ is a vector of primes that are not in the noise set. To decode, simply compute $E = X \cdot \boldsymbol{a}^{-\boldsymbol{y}} = \boldsymbol{a}^{\boldsymbol{x}-\boldsymbol{y}} \cdot e \pmod q$ and factor it using the extended Legendre theorem (i.e. Theorem 16).

Specifically, if $\boldsymbol{y} \approx \boldsymbol{x}$, then $E = U/V \pmod q$ with U and V both a product of a small number of primes. Suppose q was chosen properly (e.g. greater than cUV for some constant $c \geq 2$), then by a similar idea of the Wiener's attack against the RSA encryption scheme, we can efficiently find all the prime factors of U and V from $U/V \pmod q$ using the extended Legendre theorem. Then we can decide if $\boldsymbol{y}$ is a good fingerprint by checking if the factorizations of U and V are square-free and both are over $\boldsymbol{a}$ (except that there is precisely one factor e that is in the noise set L). Hence if $\boldsymbol{y}$ is a good fingerprint, the decoding will succeed and $\boldsymbol{y}$ will be correctly accepted. Otherwise if $\boldsymbol{y}$ is a bad fingerprint, namely it is far away from $\boldsymbol{x}$, then the decoding will fail and the checking will reject $\boldsymbol{y}$ with high probability.

Note that this secure sketch is reusable because reusing the same fingerprint $\boldsymbol{x}$ for different base vectors $\boldsymbol{a}$ and different noises e (corresponding to different devices such as cellphones, pads, notebooks, etc.) is precisely an NSP. And the fingerprint $\boldsymbol{x}$ is protected by the hardness of NSP.

A.4 Obfuscation from NSP

A similar idea can be used to obfuscate conjunctions [4, 2]. As suggested by [14, Appendix B], in order to obfuscate conjunctions, it is sufficient to obfuscate small superset functions (SSF) and big subset functions (BSF). Also by [14], an obfuscator for SSF implies an obfuscator for BSF. Hence it is sufficient to obfuscate SSF.

An SSF is a boolean function $f_{\boldsymbol{x},t} : \{0,1\}^n \to \{0,1\}; \boldsymbol{y} \mapsto f_{\boldsymbol{x},t}(\boldsymbol{y})$ with respect to a secret vector $\boldsymbol{x} \in \{0,1\}^n$ and a threshold $0 < t < n \in \mathbb{N}$ and evaluates to 1 if and only if $\boldsymbol{y}$ is a "small superset" of $\boldsymbol{x}$, i.e., the 1's in $\boldsymbol{y}$ "cover" all the 1's in $\boldsymbol{x}$ and the number of 1's in $\boldsymbol{y}$ is not greater than t.

The idea of obfuscation is similar to [14] except that we employ a noise e to make a function "reobfuscatable". We simply compute an NSP instance $(\boldsymbol{a}, X = \boldsymbol{a}^{\boldsymbol{x}} \cdot e \pmod q)$ and publish it as the obfuscated function, where $\boldsymbol{a}$ is a vector of primes that are not in the noise set L, q is a prime modulus chosen to be "just" greater than the product of $t - |\boldsymbol{x}| + 1$ primes in $\boldsymbol{a}$, and $|\boldsymbol{x}|$ is the Hamming weight of $\boldsymbol{x}$. To evaluate, first compute $E = X \cdot \boldsymbol{a}^{-\boldsymbol{y}} = \boldsymbol{a}^{\boldsymbol{x}-\boldsymbol{y}} \cdot e \pmod q$. Now if $\boldsymbol{y}$ is a "small superset" of $\boldsymbol{x}$, then $\boldsymbol{a}^{\boldsymbol{x}-\boldsymbol{y}} \cdot e$ is smaller than q and "(mod q)" can be ignored and that we have $E = \boldsymbol{a}^{\boldsymbol{x}-\boldsymbol{y}} \cdot e$. We can then recover $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{y}$ by testing divisibilities "$(a_i|E)$?" using the primes a_i in $\boldsymbol{a}$. Having $\boldsymbol{y}$ and $\boldsymbol{z}$, we can recover $\boldsymbol{x}$; and the noise e follows. In this case, $\boldsymbol{y}$ will be accepted. Otherwise, if $\boldsymbol{y}$ is not a "small superset" of $\boldsymbol{x}$, then the above decoding process will fail with high probability, meaning that we will encounter one or more of the following situations with high probability: (1) obtaining a non-binary $\boldsymbol{x}$; (2) the prime factors of E are not all in $\boldsymbol{a}$ or L; or (3) there are more than one prime factor in L. In that case, $\boldsymbol{y}$ will be rejected.

The advantage of our SSF obfuscator over the SSF obfuscator in [14] is that in our case, the same function $f_{\boldsymbol{x},t}$ can be obfuscated many times, while in the case of [14], a function $f_{\boldsymbol{x},t}$ can only be obfuscated once. Also, the conjunction obfuscation from our SSF obfuscation does not rely on any extra assumption apart from the hardness of NSP, while the conjunction obfuscation in [14, Appendix B] is based on an extra assumption which hypothesizes that the secret $\boldsymbol{x}$ is safe if it is only used twice.