Исследование FHE-based методов распознавания шаблонов

Иван Тунёв

11 июля 2022 г.

Гомоморфное шифрование: неформальное введение

- Шифрование «сохраняющее операции»
  - Но без ветвлений!
  - … медицинскими данными
  - … финансовые персональные данные
  - … энергетические датчики
- Честные тайные голосования?
- Свежее достижение → 2010 год

Гомоморфное шифрование: формально

Определение

Полностью гомоморфное шифрование *англ. Fully Homomorphic Encryption — шифрование, гомоморфное относительно любых операций над открытыми текстами. Функция шифрования* $E(k, m)$*, где* m *— это открытый текст,* k *— ключ, гомоморфна относительно операции* $*$*, если для любого ключа и открытых текстов* m_1, m_2 *существует эффективно вычислимая функция* M*, которая, принимая на вход пару* $E(k, m_1), E(k, m_2)$*, вернёт криптограмму с такую, что результатом её дешифрования будет* $m_1 * m_2$*. Аналогично вводится понятие гомоморфности относительно операции* $+$*.*

ГШ: Реально работает!

```
def demo_homomorphic_encription():
    # Инициализируем HE-объект Pyfhel
    HE = Pyfhel()
    # Контекст для заданного p
    HE.contextGen(p=65537)
    HE.keyGen()

    integer1 = 47
    integer2 = -2
    ctxt1 = HE.encryptInt(integer1)
    ctxt2 = HE.encryptInt(integer2)

    print('ctxt1=' + str(ctxt1))
    print('ctxt2=' + str(ctxt2))

    # Можно выполнять на недоверенной территории
    ctxtSum = ctxt1 + ctxt2
    ctxtSub = ctxt1 - ctxt2
    ctxtMul = ctxt1 * ctxt2

    print('Без расшифровки результаты непонятны:')
    print('ctxtSum=' + str(ctxtSum))
    print('ctxtSub=' + str(ctxtSub))
    print('ctxtMul=' + str(ctxtMul))


    resSum = HE.decryptInt(ctxtSum)
    resSub = HE.decryptInt(ctxtSub)
    resMul = HE.decryptInt(ctxtMul)

    print('И только мы, расшифровав, сможем узнать:')
    print('resSum=' + str(resSum))
    print('resSub=' + str(resSub))
    print('resMul=' + str(resMul))
```

```
ctxt1=<Pyfhel Ciphertext at 0x7fc05ee0e840, encoding=INTEGER,
size=2/2, noiseBudget=27>
ctxt2=<Pyfhel Ciphertext at 0x7fc0547d97c0, encoding=INTEGER,
size=2/2, noiseBudget=27>

Без расшифровки результаты непонятны:
ctxtSum=<Pyfhel Ciphertext at 0x7fc05386d880, encoding=INTEGER,
size=2/2, noiseBudget=27>
ctxtSub=<Pyfhel Ciphertext at 0x7fc05386d740, encoding=INTEGER,
size=2/2, noiseBudget=27>
ctxtMul=<Pyfhel Ciphertext at 0x7fc05f1ee440, encoding=INTEGER,
size=3/3, noiseBudget=1>

И только мы, расшифровав, сможем узнать:
resSum=45
resSub=49
resMul=-94
```

Гомоморфное шифрование у нас

Запускаем

- зашифрованный «алгоритм»
- на «недоверенной» территории
- и их «открытых» данных

Например:

- Поиск вирусов
  - Не раскрывая сигнатур, по которым они определяются
- Блокирование запрещенных ресурсов «черным ящиком» у провайдера
  - Не раскрывая определяющих алгоритмов

«Алгоритмы» → «Регулярные выражения и конечные автоматы»

Поставленная цель

- Реализовать алгоритм
  - *«Paperwithcode» движение*
  - Максимально компактно
  - Визуализация структур, потоков
- Получить обучающие материалы
  - Раздел в «криптографию на решетках».
  - Слайды для лекций.
- Материалы для исследований и модификаций

Изученные статьи

2020 Homomorphic Encryption for Finite Automata Genise et al.

2020 Obfuscating Finite Automata Steven D. Galbraith and Lukas Zobernig

2017 Pattern Matching on Encrypted Streams Desmoulins1 et al.

2016 Packing Messages and Optimizing Bootstrapping in GSW-FHE Hiromasa et al.

Homomorphic Encryption for Finite Automata

Nicholas Genise (Rutgers)*   Craig Gentry (Algorand Foundation)†
Shai Halevi (Algorand Foundation)†   Baiyu Li (UCSD)
Daniele Micciancio (UCSD)

March 16, 2020

Abstract

We describe a somewhat homomorphic GSW-like encryption scheme, natively encrypting matrices rather than just single elements. This scheme offers much better performance than existing homomorphic encryption schemes for evaluating encrypted (nondeterministic) finite automata (NFAs). Differently from GSW, we do not know how to reduce the security of this scheme from LWE, instead we reduce it from a stronger assumption, that can be thought of as an inhomogeneous variant of the NTRU assumption. This assumption (that we term iNTRU) may be useful and interesting in its own right, and we examine a few of its properties. We also examine methods to encode regular expressions as NFAs, and in particular explore a new optimization problem, motivated by our application to encrypted NFA evaluation. In this problem, we seek to minimize the number of states in an NFA for a given expression, subject to the constraint on the ambiguity of the NFA.

Keywords. Finite Automata, Inhomogeneous NTRU, Homomorphic Encryption, Regular Expressions.

1 Introduction

Homomorphic encryption (HE) [48] enables computation on encrypted data even without knowing the secret key. Ten years after Gentry described the first scheme capable of supporting arbitrary computations [23], we now have an arsenal of several different schemes and variations, with various capabilities and tradeoffs (see, e.g., [52, 12, 11, 39, 21, 26, 17] for a few examples).

Our original motivation for the current work is the simple example of encrypted virus scan: consider a center that deploys many remote systems, operating in many different environments, and wants to protect them against viruses that it knows about. The center would like to periodically send updated virus signatures to all its systems, and have them scan their systems to check for infections. The virus signatures, however, could be sensitive, perhaps because some of them are not yet widely known and exposing the signatures could tip the hand of the center as it develops countermeasures.

*This work was done when the author was at UCSD

†This work was done when the authors were in IBM Research

A plausible solution would have the center encrypt the virus signatures, the remote systems could then perform the virus scan on the encrypted signatures, and report the (encrypted) results to the center. The center could then decrypt, and take appropriate actions when infections are detected. As virus signatures usually take the form of many small regular expressions[1], this application calls for a homomorphic encryption scheme that can quickly test for a match against many small regular expressions. Equivalently, it should quickly evaluate (many, encrypted) nondeterministic finite automata (NFAs) on a given cleartext file. Notice that this is quite different from, and incomparable to, the DFA computation problem studied in previous works on homomorphic encryption, like [22, 18, 19]. Specifically, nondeterminism aside, the crucial difference is that those works consider the evaluation of a plaintext automaton on an encrypted file. In other words, the roles of the input and the program are reversed. In our motivating application, the problem studied in [22, 18, 19] would correspond to searching for arbitrary (possibly nonregular) patterns, on files described by regular languages, a very unlikely scenario.

Evaluating an encrypted NFA on a cleartext string $w = w_1 \cdots w_k$ can be done by computing a product of a single vector $\mathbf{v}$ (representing the initial state of the NFA) by many matrices $\mathbf{M}_{w_i}$ (representing the transition matrices of the NFA associated to each input symbol w_i). Namely the operation that we want to support is computing

$$\mathbf{u} \coloneqq \left(\prod_{i=k}^{1} \mathbf{M}_{w_i}\right) \times \mathbf{v},$$

(with operations over the integers), where the matrices $\mathbf{M}_{w_i}$ and the vector $\mathbf{v}$ are encrypted.[2] Most of the HE schemes from above can be used to carry out this computation, but none of them is ideal for the job. For practical purposes, the homomorphic schemes that offer the best performance are either the BGV-type schemes (scale-invariant or not), or GSW-type schemes.

BGV-type schemes. These schemes have an advantage that they can use *packed ciphertexts*, where each ciphertext encrypts not just one plaintext element but a vector of them, and each ciphertext operation affects all the elements of the vector simultaneously, cf. [51]. Moreover, they can even be made to support efficient matrix-vector operations, as was demonstrated in [27].[3]

However, for BGV-type schemes it is crucial to keep the computation multiplicative depth to a minimum, which in our case means using a binary multiplication tree. But this means that we have to use matrix-matrix multiplication[4] (rather than the matrix-vector products that are computed in the sequential procedure). This increases the total work (and hence the computation time) by a factor equal to the dimension of these matrices — which must be substantial for security reasons.

[1]For example, many ClamAV virus signatures (`https://www.clamav.net/downloads`) are regular expressions of the form $\Sigma^* K_1 \cdots \Sigma^* K_n \cdot \Sigma^*$ with no more than 1K symbols, where Σ is the alphabet and each K_i is a set of a few hex strings.

[2]The initial vector $\mathbf{v}$ is not required to be encrypted, as it reveals no information about the automaton. However, the intermediate vectors obtained after each matrix-vector multiplication should be kept secret. So, we will need a scheme supporting matrix-vector multiplication where both the matrix and the vector are encrypted.

[3]The techniques in [27] only handle multiplication of plaintext matrices by encrypted vectors, but many of these tools can be adapted to the case of encrypted matrices.

[4]Technically, the nodes on the rightmost path of the tree can use matrix-vector multiplications, but this makes hardly any difference on the efficiency of the overall computation.

GSW-type schemes. A major advantage of GSW-like schemes is the asymmetric noise growth, that makes it possible to handle sequential processing of products [14]. For our purposes, it lets us evaluate the product while performing only matrix-vector multiplications.

While "textbook GSW" can only encrypt individual elements, it is possible to adapt the ciphertext-packing techniques from [51] also to GSW, as long as we have a priori bound on the size of the plaintext vectors that occur in the computation. However porting the matrix-multiplication optimizations from [27] is far from simple, and we expect significant overhead when trying to implement it in practice.

In [29], Hiromasa, Abe, and Okamoto proposed a GSW-like FHE scheme that is capable of encrypting square matrices and doing homomorphic matrix addition and multiplication. The HAO15 FHE scheme can be viewed as a matrix extension of the standard GSW-FHE scheme, where the secret key $\mathbf{S} = [\mathbf{I}| - \mathbf{S}']$ consists of a random secret matrix $\mathbf{S}'$. Like in GSW [26], the decryption invariant for a ciphertext $\mathbf{C}$ encrypting a message $\mathbf{M}$ relative to the secret key $\mathbf{S}$ is

$$\mathbf{S} \times \mathbf{C} = \mathbf{M} \times \mathbf{S} \times \mathbf{G} + \mathbf{E} \pmod q,$$

where $\mathbf{E}$ is a low-norm error and $\mathbf{G}$ is the "gadget matrix" from [43]. Notice that $\mathbf{M}$ and $\mathbf{S}$ are both matrices in the matrix-FHE case, whereas in the GSW scheme $\mathbf{M}$ is a scalar and $\mathbf{S}$ is a vector. The GSW security reduction [26] from the learning-with-errors (LWE) problem still applies to the HAO15 scheme, except that an additional circular security assumption is required. Being able to encrypt matrices in an atomic operation and support homomorphic matrix operations makes the HAO15 scheme an interesting candidate to use in our application of homomorphic NFA evaluation. Moreover, as we will show in Section 3.1, the HAO15 scheme with some modification can also encrypt vectors and homomorphically multiply an encrypted matrix by an encrypted vector. However, the HAO15 scheme is not optimal due to overhead in the size of keys and ciphertexts. So we seek to find a better solution that would allow us to scan longer strings with faster execution times in practice.

1.1 Our New HE Scheme

In this work we introduce a new scheme, that can be viewed as another GSW-type encryption for matrices but with a different hardness assumption. (Alternatively, it can be viewed as a variant of the GGH15 graded encoding [24], but with no zero-test parameter.) In addition, our scheme can also encrypt vectors and natively support homomorphic matrix-vector multiplication. Similar to the HAO15 scheme, the decryption invariant in our scheme for a ciphertext $\mathbf{C} \leftarrow \mathsf{Enc}_{\mathbf{S}}(\mathbf{M})$ encrypting a matrix $\mathbf{M}$ is also $\mathbf{S} \times \mathbf{C} = \mathbf{MSG} + \mathbf{E} \pmod q$, where $\mathbf{E}$ is a low-norm error matrix.[5] Differently from the HAO15 scheme, in our construction we assume that the key $\mathbf{S}$ is a square invertible matrix, and so we can express the ciphertext as $\mathbf{C} := \mathbf{S}^{-1}(\mathbf{M} \times \mathbf{S} \times \mathbf{G} + \mathbf{E}) \bmod q$. As a result, both keys and ciphertexts are smaller in our scheme.

The operations of the scheme, and the analysis of the noise development are identical to the GSW scheme, except that here we typically cannot ensure that the plaintext size never grows, and instead must use properties of the application to reason about the plaintext size.

When it comes to security, however, we can no longer use the GSW reduction [26] from the LWE problem. That reduction relies heavily on the scalar $\mathbf{M}$ commuting with the vector $\mathbf{S}$, which no longer holds in our case. Instead, we reduce the security of this scheme from a stronger assumption,

[5] As we describe later, we use a slightly different variant to encrypt the vector $\mathbf{v}$.

that can be viewed as an inhomogeneous version of NTRU (or alternatively as an LWE instance with an additional hint).

1.2 The iNTRU Hardness Assumption

Recall that in LWE[6], we are given two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$ $(m > n)$, with $\mathbf{A}$ a uniformly random matrix, and need to decide if $\mathbf{B}$ is also a uniformly random matrix, or it is chosen as $\mathbf{B} = \mathbf{SA} + \mathbf{E}$ with a uniform $\mathbf{S} \in \mathbb{Z}_q^{n \times n}$ and a low-norm $\mathbf{E} \in \mathbb{Z}_q^{n \times m}$.

It is easy to see that this problem becomes easy if we are also given a trapdoor for the matrix $\mathbf{A}$, in this case it is even easy to recover the secret matrix $\mathbf{S}$ when $\mathbf{B} = \mathbf{SA} + \mathbf{E}$. But what if we are given a trapdoor for the matrix $\mathbf{B}$ instead? In this case we do not know of any effective distinguisher, so we assume that the decision problem is still hard and show a hardness reduction from this version of LWE to our hardness assumption, iNTRU, in Section 4. We remark that this "LWE with a trapdoor for $\mathbf{B}$" assumption is not standard and it deserves further study.

Once we know a trapdoor for $\mathbf{B}$, we might as well consider the case where $\mathbf{B}$ is the gadget matrix $\mathbf{G}$ (for which everyone knows a trapdoor). Namely we assume that the following decision problem is hard:

iNTRU. As in LWE, we have the parameters n, m, q, with $m > n \log q$ and $q > m$. The input is a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, which is either uniform in $\mathbb{Z}_q^{n \times m}$, or is set as $\mathbf{A} := \mathbf{S}^{-1}(\mathbf{G} - \mathbf{E}) \bmod q$ (with $\mathbf{S} \in \mathbb{Z}_q^{n \times n}$ a random invertible matrix, $\mathbf{G}$ the gadget matrix, and $\mathbf{E}$ a low-norm matrix). The goal is to decide which is the case.

One can think of the above problem as an inhomogeneous version of NTRU, over matrices, as follows. Recall that in the NTRU cryptosystem [30], the secret key is given by two polynomials (or ring elements) with small coefficients f, g, and the corresponding public key is the product $h = f^{-1} \cdot g$. The NTRU cryptosystem can be proved secure under the assumption that this public key h is pseudorandom, i.e., indistinguishable from a uniformly random polynomial (or ring element) with arbitrary coefficients. We extend this assumption as follows. First, we replace g with a sequence of vectors $g_1, \ldots, g_k$, chosen independently at random, with small coefficients. Then, the assumption is that $f^{-1}g_1, f^{-1}g_2, \ldots, f^{-1}g_k$ is pseudorandom. This is a simple syntactic extension of NTRU (that would allow, for example, the encryption of longer messages), akin to changing some parameter, and not a qualitative change in the security assumption. Next, we add a (known, constant) "shift", replacing each g_i with $(2^{i-1} - g_i)$, and still requiring $f^{-1}(1 - g_1), f^{-1}(2 - g_2), \ldots, f^{-1}(2^{k-1} - g_k)$ to be indistinguishable from uniform. We call this the "inhomogeneous" NTRU assumption. Finally, instead of working over a ring of polynomials of degree n, we replace each $f, g_1, \ldots, g_k$ with a square $n \times n$ random matrix with small entries. Intuitively, moving from polynomial rings (which are commutative) to the ring of matrices, should only make the assumption weaker, though we do not know how to prove a formal relation between the two problems. This last problem is essentially equivalent to the pseudorandomness of $\mathbf{A} = \mathbf{S}^{-1}(\mathbf{G} - \mathbf{E})$, where $\mathbf{E} = [\mathbf{E}_0 | \ldots | \mathbf{E}_k]$ is a random matrix with small entries, and $\mathbf{G} = [\mathbf{0} | \mathbf{I} | 2\mathbf{I} | \ldots | 2^{k-1}\mathbf{I}]$ is a constant known matrix. In fact, putting $\mathbf{A}$ in Hermite Normal Form [42] "cancels out" the $\mathbf{S}$ matrix, and gives a sequence of square matrices $-\mathbf{E}_0^{-1}(2\mathbf{I}^{i-1} - \mathbf{E}_i)$, corresponding to the matrix version of our inhomogeneous NTRU problem[7] with $f = -\mathbf{E}_0$ and $g_i = \mathbf{E}_i$.

[6] Here we refer to the multiple-secret variant of LWE, which can be reduced from the normal LWE.

[7] Matrix-NTRU has been used in lattice-based signatures [6], though the most efficient versions of these lattice signatures use the standard, algebraic NTRU assumption.

1.3 From Regular Expression to NFAs

While our scheme directly supports the evaluation of (encrypted) NFAs, patterns (e.g., virus signatures) are typically, and most conveniently, represented by regular expressions. Since the noise growth of our homomorphic encryption scheme depends on the details of the NFA being evaluated and its computations, the conversion of regular expressions to NFA is a critical part of our application. In Section 5 we describe a specific conversion following the method of [15, 4] based on the use of *partial derivatives* of regular expressions, which is both very elegant and efficient. Derivatives of regular expressions [15] are themselves regular expressions and they are defined similarly to formal derivatives of arithmetic expressions, e.g., $\mathrm{d}_a(e_0 + e_1) = \mathrm{d}_a(e_0) + \mathrm{d}_a(e_1)$ for the sum (set union) operation, and $\mathrm{d}_a(e^*) = \mathrm{d}_a(e)e^*$ for exponentiation (Kleene star). Informally, when parsing an input string according to regular expression e, the derivative $\mathrm{d}_a(e)$ represents the part of the input to be expected after reading a first symbol "a". A regular expression e can be converted into an automaton with states labeled by derivatives (modulo a natural equivalence relation on regular expressions), and transitions of the form $e \xrightarrow{a} \mathrm{d}_a(e)$. A classical result of Brzozowski [15] shows that this produces an automaton with a finite number of states, and, in fact, the minimal DFA of the regular expression. As our homomorphic encryption scheme supports the evaluation of nondeterministic automata, we are interested in the conversion of regular expressions to NFAs, which are potentially much smaller than the equivalent minimal DFA. However, optimizing NFAs in our application is far from trivial. To start with, in stark contrast to the DFA case, minimizing the number of states of an NFA is a PSPACE-complete problem. Moreover, due to noise growth, minimizing the number of states may not even be the right goal for our homomorphic encryption application. We address the first issue by using the *partial derivative* construction of [4], where a partial derivative $\partial_a(e)$ maps an expression e to a *set* of regular expressions (representing possible nondeterministic choices), and in particular $\partial_a(e_0 + e_1) = \partial_a(e_0) \cup \partial_a(e_1)$. This construction results in NFAs that, while not necessarily minimal, have a very small number of states, bounded by the number of alphabet symbols in the input regular expression. In order to bound the noise growth, we show that a simple optimization of the homomorphic NFA evaluation procedure[8] allows to relate the noise growth to the *degree of ambiguity* of the NFA, a standard quantity studied in automata theory, which can be evaluated in polynomial time [54]. We reduce the problem of finding an optimal noise to a variant of NFA minimization problem with bounded ambiguity. Although solving this optimization problem is hard in general, we use techniques of determining ambiguity in Section 5 to explore some tradeoffs between automata size and degree of ambiguity/noise growth.

1.4 Implementation and Performance

We implemented our scheme in C++ using the Number Theory Library (NTL) and describe its details in Section 6. Despite being a simple implementation without optimizations, the on-line pattern matching was exceptionally fast. For example, we could homomorphically match a 65536 bit string in 394 seconds on an encrypted NFA with 1024 states of size 66Mb. Using the same set of parameters, we estimate that an HAO15 implementation can only match up to 16000 bits with a slower execution time and a bigger program size. More performance details and comparisons can be found in Section 6.

[8] Namely, one can let the initial state vector be an "errorless" encryption, because the initial state does not reveal any information about the rest of the automaton.

1.5 Related Work

As already mentioned, the problem of homomorphically evaluating finite automata or branching programs has been considered before [14, 22, 18, 19], but in a very different context, where the branching program or automaton are publicly known, and the computation is performed homomorphically on an encrypted input string. This is motivated, for example, by applications to FHE bootstrapping, where the program is specified by the publicly known decryption/refreshing procedure, and the input in the (encrypted) secret key. In our setting, the role of the program and input are reversed, and we want the computation to be homomorphic on the automaton, rather than the input string. In the case of general computation, program and input are easily interchanged using a universal Turing machine. But in the case of restricted models of computation, like finite automata, swapping the program and the inputs results in a completely different problem.

On the relation with other matrix-FHE schemes. As we mentioned earlier, the HAO15 [29] FHE scheme is also capable of encrypting square matrices and doing homomorphic matrix addition and multiplication on ciphertexts. In the private-key version of their scheme, the secret key is $\mathbf{S} = [\mathbf{I}_r - \mathbf{S}']$ for a secret matrix $\mathbf{S}'$, and a matrix $\mathbf{M} \in \mathbb{Z}^{r \times r}$ is encrypted as

$$\mathbf{C} = \left(\frac{\mathbf{S}'\mathbf{A} + \mathbf{E}}{\mathbf{A}}\right) + \left(\frac{\mathbf{MS}}{\mathbf{0}}\right) \times \mathbf{G} \bmod q,$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times N}$, $\mathbf{E} \leftarrow \chi^{r \times N}$ for $N = (n + r)\lceil \log q \rceil$.

It may be tempting to claim that our scheme is the same as the HAO15 scheme due to having the same decryption invariant $\mathbf{SC} = \mathbf{MSG} + \mathbf{E}$. However, these two schemes are not quite identical. The relation between them is very similar to the relation between NTRU and RLWE Regev-like schemes[9], where the difference is that the secret key $\mathbf{S}$ is a small square matrix for NTRU (representing multiply-by-s in the ring), whereas the secret key is $\mathbf{S} = [\mathbf{I}|\mathbf{S}']$ in RLWE (where $\mathbf{S}'$ represents multiply-by-s' in the ring). Notice that, instead of the Regev invariant, both the HAO15 scheme and our scheme use the GSW-like invariant $\mathbf{SC} = \mathbf{MSG} + \mathbf{E}$ for a small noise matrix $\mathbf{E}$.

More specifically, in our scheme the secret key $\mathbf{S}$ is a small square matrix that must be invertible, while in HAO15 we have $\mathbf{S} = [\mathbf{I} - \mathbf{S}']$ where $\mathbf{S}'$ can be any random matrix. Consider the "leveled versions" of the HAO15 scheme and our scheme, in which the secret key matrices $\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_L$ are generated such that $\mathbf{S}_i$ is used to encrypt the matrices in level i of the computation. In both schemes it holds that

$$\mathbf{S}_i\mathbf{C}_i = \mathbf{MS}_{i+1}\mathbf{G} + \mathbf{E}_i.$$

The security of the HAO15 scheme can be reduced from the standard LWE assumption, while our scheme relies on the NTRU-like assumption that we introduce. On the other hand, our scheme is more efficient: we encrypt a matrix $\mathbf{M} \in \mathbb{Z}_q^{r \times r}$ in a ciphertext matrix of dimension $\max(r, \lambda)$, whereas the HAO15 scheme requires a dimension $r + \lambda$ ciphertext matrix. One can view our scheme as an NTRU-like variant of the HAO15 scheme (or perhaps an NTRU-like variant of the GSW scheme). From that viewpoint, we introduce in this work the assumption that lets us adapt NTRU to get a GSW-like scheme.

[9] Consider writing both NTRU and RLWE-Regev in matrix form, representing ring elements by their matrices: In both NTRU and RLWE-Regev we have a ciphertext matrix $\mathbf{C}$ encrypting a plaintext matrix $\mathbf{M}$ relative to the secret matrix $\mathbf{S}$ (and plaintext space mod p) if $\mathbf{SC} = \mathbf{M} + p\mathbf{E} \bmod q$.

When applied to homomorphically evaluating NFAs, the efficiency advantage of our scheme is more significant. Note that the HAO15 scheme can be used to do homomorphic matrix-vector multiplication as well. But, since we rely on an NTRU-like assumption, the noise bound in our scheme is smaller than the noise bound in the HAO15 scheme, which allows us to homomorphically evaluate longer strings with the same lattice parameters. In terms of the complexity of the homomorphic computation on encrypted NFAs, our scheme runs faster than the HAO15 scheme in practice due to smaller ciphertexts. For more detailed performance comparison, we refer the readers to Section 6 and Appendix C.

Recently, Wang et. al. [53] proposed another matrix-FHE scheme, similar to [11], that has smaller ciphertexts than the HAO15 scheme and can be reduced from the standard LWE assumption. We note that it is possible to perform homomorphic matrix-vector multiplication in their scheme. However, their scheme relies heavily on tensor product to perform homomorphic multiplication, so the security and the complexity of applying their scheme to homomorphic NFA computation is at least on the same level as the HAO15 scheme.

2 Preliminaries

We denote vectors by lower-case bold letters (e.g., $\mathbf{v}$), and we assume they are always in column form. We denote matrices by upper-case bold letters (e.g., $\mathbf{M}$). A distribution $\mathcal{D}$ over a finite set X is ϵ-uniform if its statistical distance from the uniform distribution over X is at most ϵ, where the statistical difference between two distributions $\mathcal{D}_1, \mathcal{D}_2$ over a finite domain X is $\frac{1}{2}\sum_{x \in X} |\mathcal{D}_1(x) - \mathcal{D}_2(x)|$. We denote by $x \leftarrow \mathcal{D}$ drawing x from the distribution $\mathcal{D}$, and for a set X we denote by $x \leftarrow X$ drawing x uniformly at random from X.

2.1 Leftover Hash Lemma

A distribution $\mathcal{D}$ over X has min-entropy k if $\max_{x \in X} \mathcal{D}(x) = 2^{-k}$. A family $\mathcal{H}$ of hash functions from X to Y (with Y a finite set) is said to be 2-universal if for all distinct $x, x' \in X$, $\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')] = 1/|Y|$.

Lemma 2.1. *(Leftover Hash Lemma [28]). Let $\mathcal{H}$ be a family of 2-universal hash functions from X to Y, and let $\mathcal{D}$ be a distribution over X with min-entropy k. Suppose that $h \leftarrow \mathcal{H}$ and $x \leftarrow \mathcal{D}$ are chosen independently, then, $(h, h(x))$ is $(\frac{1}{2}\sqrt{|Y|/2^k})$-uniform over $\mathcal{H} \times Y$.*

In this work we apply Lemma 2.1 to the hashing family $\mathcal{H} : \mathbb{Z}_q^m \to \mathbb{Z}_q^n$ defined by

$$\mathcal{H} = \{h_A(\mathbf{v}) = \mathbf{Av} \bmod q\}_{\mathbf{A} \in \mathbb{Z}_q^{n \times m}},$$

(which is clearly 2-universal). In particular we use the following corollary:

Corollary 2.2. *Fix the integers k, n, m, m', q, and let $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m$ be independent distributions over $\mathbb{Z}_q^m$, all with min-entropy at least k. Let $\mathcal{D}$ be a distribution over matrices $\mathbf{R} \in \mathbb{Z}_q^{m \times m'}$, where the i'th column is drawn from $\mathcal{D}_i$. Then the distribution*

$$\{(\mathbf{A}, \mathbf{AR} \bmod q) : \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{R} \leftarrow \mathcal{D}\}$$

is $(\frac{m'}{2}\sqrt{q^n/2^k})$-uniform over $\mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^{n \times m'}$.

2.2 Gadget Lattice Sampling

Definitions. We consider the norm of a matrix as the length of its longest column in the l_2 norm. A lattice Λ is a discrete subgroup of $\mathbb{R}^n$ (we only consider full-rank, integer lattices). It can be represented by a basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$ where the lattice is the set of all integer combinations of $\mathbf{B}$'s columns. Let $\mathbf{G} = [\mathbf{I}|2\mathbf{I}|\cdots|2^{\ell-1}\mathbf{I}] \in \mathbb{Z}_q^{n \times n\ell}$ where $\ell = \lceil \log_2(q) \rceil$. The G-lattice for a fixed modulus q is $\Lambda_q^{\perp}(\mathbf{G}) = \{\mathbf{x} \in \mathbb{Z}^{n\ell} : \mathbf{Gx} \bmod q = \mathbf{0}\}$. The distribution sampled over $\Lambda_q^{\perp}(\mathbf{G})$ and its integer cosets is the discrete gaussian, a gaussian distribution conditioned on being in the lattice. The probability a sample equals some lattice coset vector $\mathbf{y}$ is proportional to $\exp(-\pi\|\mathbf{y}\|^2/s^2)$ where $s > 0$ is the width of the gaussian (we are only concerned with $\mathbf{0}$-centered distributions). Denote a discrete gaussian of width s on a lattice coset $\Lambda + \mathbf{c}$ as $\mathcal{D}_{\Lambda+\mathbf{c},s}$. We can efficiently sample from $\mathcal{D}_{\Lambda_q^{\perp}(\mathbf{G})+\mathbf{v},s}$ for any $q \geq 2$ and $s \geq \sqrt{5\ln(2n\ell+4)/\pi}$ (Theorem 4.1 [43] and Lemma 2.3 [13]). We denote $\mathbf{G}^{-1}(\mathbf{v})$ as a discrete gaussian vector $\mathbf{y}$ such that $\mathbf{Gy} = \mathbf{v} \bmod q$. Further, we assume the width is set just above twice the smoothing parameter (defined below) of the G-lattice.

Concentration and min-entropy. The smoothing parameter [44] of a lattice is needed for our purposes, and it is denoted as $\eta_\epsilon(\Lambda)$ for an $\epsilon > 0$. Informally, this is the smallest width for which a discrete gaussian shares many properties of the continuous gaussian distribution. If $\mathbf{B}$ is a basis with minimum Gram-Schmidt norm $\|\widetilde{\mathbf{B}}\|$, we can bound the smoothing parameter $\eta_\epsilon(\Lambda) \leq \|\widetilde{\mathbf{B}}\|\omega(\sqrt{\log n})$ for negligible $\epsilon(n) = n^{-\omega(1)}$ [25]. Discrete gaussian samples' l_2 norms are bounded by their width as follows.

Lemma 2.3. *(Lemma 1.5 [7]) Let $\Lambda \subset \mathbb{R}^n$ be a lattice, $r \geq \eta_\epsilon(\Lambda)$ for some $\epsilon \in (0,1)$, and $c \in \mathbb{R}^n$. Then,*

$$\Pr(\|\mathcal{D}_{\Lambda+\mathbf{c},r} \geq r\sqrt{n}\|) \leq 2^{-n} \cdot \left(\frac{1+\epsilon}{1-\epsilon}\right).$$

Therefore, we can efficiently sample a discrete gaussian $\mathbf{G}^{-1}(\cdot)$ with length less than $\widetilde{O}(\sqrt{n \log q})$[10] with overwhelming probability, and assume $\mathbf{G}^{-1}(\cdot)$'s support is $\mathbb{Z}_q^{n\ell}$. Since we will be using the leftover hash lemma on discrete gaussian input, we will use the following lemma on the min-entropy of a discrete gaussian. Further, the proof of Lemma 2.4 is identical to the proof of [46, Lemma 2.11].

Lemma 2.4. *(Lemma 2.11 [46]) Let $\Lambda + \mathbf{v} \subset \mathbb{R}^n$ be a lattice coset, $c > 0$, and $s \geq 2^{1+c}\eta_\epsilon(\Lambda)$ for $\epsilon \in (0,1)$. Then for any $\mathbf{y} \in \Lambda + \mathbf{v}$ and for $\mathbf{x} \leftarrow \mathcal{D}_{\Lambda+\mathbf{v},s}$,*

$$\Pr(\mathbf{x} = \mathbf{y}) \leq 2^{-n(1+c)}\left(\frac{1+\epsilon}{1-\epsilon}\right).$$

Leftover Hash Lemma with $\mathbf{G}^{-1}(\cdot)$. Let $m = n\ell$, now we can replace the distributions $\mathcal{D}_i$ in Corollary 2.2 with independent discrete gaussian samples $\mathbf{G}^{-1}(\mathbf{v})$ (with potential repeats in the coset vector $\mathbf{v}$). Let $\mathbf{R} \leftarrow \mathbf{G}^{-1}(\mathbf{X})$ in Corollary 2.2 for some $\mathbf{X} \in \mathbb{Z}_q^{n \times m'}$ with $\mathbf{R}$'s columns sampled independently. Then by the lemmas above, the min-entropy a column of $\mathbf{R}$ is at least $n(1+c)\log q - 2$ whenever $\mathbf{G}^{-1}(\cdot)$'s width is just above twice $\eta_\epsilon(\Lambda_q^{\perp}(\mathbf{G}))$ for any $\epsilon \in (0, 1/2]$. Say we let $c = \log_q(2)$ in Lemma 2.4. This implies the distribution

$$\{(\mathbf{A}, \mathbf{AR} \bmod q) : \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{R} \leftarrow \mathbf{G}^{-1}(\mathbf{X})\}$$

[10] $\widetilde{O}(\cdot)$ hides poly-logarithmic factors in n.

is $O(m'2^{-n/2})$-uniform for any $\mathbf{X} \in \mathbb{Z}_q^{m \times m'}$.

3 The Schemes

Given an NFA $\mathcal{M}$ of r states over a finite alphabet Σ, we denote by $\mathbf{M}_\sigma \in \{0,1\}^{r \times r}$ the transition matrix of $\mathcal{M}$ for each input symbol $\sigma \in \Sigma$, where $(\mathbf{M}_\sigma)_{j,i} = 1$ if and only if there is a transition from state i to state j on σ. Let $\mathbf{v} \in \{0,1\}^r$ be the vector representing the initial states. To check if a string $w = w_1 \cdots w_k \in \Sigma^*$ is accepted by $\mathcal{M}$, we simply check whether there are any non-zero entries in the vector $(\prod_{i=k}^{1} \mathbf{M}_{w_i}) \times \mathbf{v}$ that correspond to final states. So we need a scheme that can compute matrix-vector multiplication homomorphically over encrypted matrices and vectors.

3.1 The HAO15 matrix-FHE scheme [29]

The FHE scheme from [29] can be extended to support homomorphic matrix-vector multiplication. We first recall the private-key version of the HAO15 scheme, and we then slightly extend it for vector encryption and homomorphic matrix-vector multiplication. For a given security parameter λ, choose lattice parameters n, m, q and a noise distribution χ over $\mathbb{Z}_q$. Let $\ell = \lceil \log q \rceil$, $m = (n + r)\log q$, and $N = (n + r)\ell$. Here we describe a leveled version of the HAO15 scheme that supports multiplication depth up to $k \geq 1$. We abuse notation and have $\mathbf{G} = [\mathbf{0}|\mathbf{I}|2\mathbf{I}|\cdots|2^{\ell-1}\mathbf{I}]$ in this subsection.

Key generation. Same as in HAO15, the secret key for level $i \geq 0$ is set to $\mathsf{sk}_i := \mathbf{S}_i = [\mathbf{I}_r| - \mathbf{S}'_i]$, where $\mathbf{S}'_i \leftarrow \chi^{r \times n}$.

Matrix encryption. Given a plaintext matrix $\mathbf{M} \in \{0,1\}^{r \times r}$ and a level $i \geq 0$, to encrypt it for the i'th level of computation, the HAO15 scheme outputs

$$\mathbf{C} := \mathsf{HAO.MatEnc}_{\mathsf{sk}_i}(M) = \left(\frac{\mathbf{S}'_i\mathbf{A}' + \mathbf{E}}{\mathbf{A}'}\right) + \left(\frac{\mathbf{MS}_{i-1}}{\mathbf{0}_{n \times (n+r)}}\right)\mathbf{G} \bmod q,$$

where $\mathbf{A}' \leftarrow \mathbb{Z}_q^{n \times N}$ and $\mathbf{E} \leftarrow \chi^{r \times N}$. For $i = 0$, we consider $\mathbf{S}_{-1} = [\mathbf{I}_r|\mathbf{0}_{r \times n}]$. Notice that $\mathbf{C} \in \mathbb{Z}_q^{(r+n) \times N}$. The decryption procedure is exactly the same as in [29], but we skip it as it is not needed in our application.

Vector encryption and decryption. For a vector $\mathbf{v} \in \mathbb{Z}_q^r$, we can follow the same idea as in the matrix encryption procedure, except that we do not multiply $\mathbf{v}$ by $\mathbf{S}$ nor $\mathbf{G}$. Since we only need to encrypt the initial state vector to evaluate an NFA, we always encrypt a vector using the secret key for the first level:

$$\mathbf{c} := \mathsf{HAO.VecEnc}_{\mathsf{sk}_0}(\mathbf{v}) = \left(\frac{\mathbf{S}'_0\mathbf{a} + \mathbf{e}}{\mathbf{a}}\right) + \left(\frac{\mathbf{v}}{\mathbf{0}_n}\right) \bmod q,$$

where $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ and $\mathbf{e} \leftarrow \chi^r$. Note that $\mathbf{c}$ has dimension $r + n$. To decrypt a ciphertext vector $\mathbf{c}$ from the i'th level of a computation, output the vector

$$\mathbf{v}' := \mathsf{HAO.VecDec}_{\mathsf{sk}_i}(\mathbf{c}) = \lceil \mathbf{S}_i\mathbf{c} \rfloor_2.$$

Homomorphic operations. To add and multiply two ciphertext matrices $\mathbf{C}_1$ and $\mathbf{C}_2$, we follow [29]: $\mathsf{HAO.Add}(\mathbf{C}_1, \mathbf{C}_2) = \mathbf{C}_1 + \mathbf{C}_2$, and $\mathsf{HAO.Mul}(\mathbf{C}_1, \mathbf{C}_2) = \mathbf{C}_1 \times \mathbf{G}^{-1}(\mathbf{C}_2)$. To multiply a ciphertext matrix $\mathbf{C}$ by an encrypted vector $\mathbf{c}$, output

$$\mathsf{HAO.Mul}(\mathbf{C}, \mathbf{c}) := \mathbf{C} \times \mathbf{G}^{-1}(\mathbf{c}).$$

The security of this extended scheme can be proved in the same way as in [29], reducing from the standard $\mathrm{DLWE}_{n,m,q,\chi}$ hardness assumption. It is easy to check that, if $\mathbf{C}$ is an encryption of $\mathbf{M} \in \{0,1\}^{r \times r}$ for level i and $\mathbf{c}$ is an encryption of $\mathbf{v}$ of level $i-1$, then $\mathbf{S}_i \times (\mathbf{C} \times \mathbf{G}^{-1}(\mathbf{c})) = \mathbf{Mv} + \mathbf{e}'$ for some low norm error vector $\mathbf{e}'$. More generally, for any $\mathbf{M}_i \in \{0,1\}^{r \times r}$ for $i = 1, \ldots, k$ and $\mathbf{v} \in \mathbb{Z}_q^r$, if $\mathbf{C}_i \leftarrow \mathsf{HAO.MatEnc}_{\mathsf{sk}_i}(\mathbf{M}_i)$ with an error matrix $\mathbf{E}_i$ for each i, $\mathbf{c}_0 \leftarrow \mathsf{HAO.VecEnc}_{\mathsf{sk}_0}(\mathbf{v})$ with an error vector $\mathbf{e}$, and $\mathbf{c}_i \leftarrow \mathsf{HAO.Mul}(\mathbf{C}_i, \mathbf{c}_{i-1})$ for $i = 1, \ldots, k$, then $\mathbf{S}_k \times \mathbf{c}_k = (\prod_{j=k}^{1} \mathbf{M}_j)\mathbf{v} + \mathbf{e}_k$ where

$$\mathbf{e}_k = \mathbf{E}_k\mathbf{G}^{-1}(\mathbf{c}_{k-1}) + \sum_{i=2}^{k}(\prod_{j=k}^{i} \mathbf{M}_j)\mathbf{E}_{i-1}\mathbf{G}^{-1}(\mathbf{c}_{i-2}) + (\prod_{j=k}^{1} \mathbf{M}_j)\mathbf{e}.$$

The l_∞ norm of $\mathbf{e}_k$ can be bounded by

$$\|\mathbf{e}_k\|_\infty \leq \chi N(1 + k \max_{1 \leq i \leq k} \|\prod_{j=k}^{i} \mathbf{M}_j\|_\infty). \qquad (1)$$

To successfully decrypt $\mathbf{c}_k$, we require $\|\mathbf{e}_k\|_\infty \leq q/8$ as in [29].

3.2 Our new matrix-HE scheme

To achieve sufficient level of security and a desired capability of homomorphic NFA evaluation, we may need to use a large lattice dimension n in practice. The above extension of the HAO15 scheme seems suboptimal with an overhead n in ciphertext dimension. In this section we describe a new matrix homomorphic encryption scheme that supports atomic matrix and vector encryption and matrix-vector multiplication. Our scheme is more efficient in practical applications.

Fix integer parameters n, m, q (to be determined later) and an error distribution χ over $\mathbb{Z}_q$ that outputs with high probability integers of magnitude $\ll q$. Given any NFA with $r \leq n$ states, we pad its transition matrices $\mathbf{M}_\sigma$ with 0 entries such that $\mathbf{M}_\sigma \in \{0,1\}^{n \times n}$ for all $\sigma \in \Sigma$. For our application we use two variants of (private-key) encryption, one for matrices and the other for vectors. Both variants share a noise-sampling procedure, that takes as input the secret key and another vector (that comes from the plaintext) and outputs a noise vector for use in the encryption (which may be different than just sampling from χ). We denote this procedure by $\mathbf{e} \leftarrow \mathsf{NoiseSamp}(\mathsf{sk}, \mathbf{v})$, and will describe it later in this section.

Key generation. We draw two matrices using χ, a square matrix $\mathbf{S} \leftarrow \chi^{n \times n}$ and a rectangular $\mathbf{E} \leftarrow \chi^{n \times m}$ (which is only used in the $\mathsf{NoiseSamp}$ procedure). We insist that $\mathbf{S}$ is invertible, and re-sample if it is not (which happens with a small probability $\approx 1/q$). The secret key is $\mathsf{sk} := (\mathbf{S}, \mathbf{E})$.

The $\mathsf{NoiseSamp}$ procedure. To prove semantic security of our encryption method, we need a somewhat convoluted procedure for sampling the noise. Specifically, the procedure $\mathsf{NoiseSamp}((\mathbf{S}, \mathbf{E}), \mathbf{v})$ begins by sampling $\mathbf{r} \leftarrow \mathbf{G}^{-1}(\mathbf{v})$, then outputs $\mathbf{e} := \mathbf{E} \times \mathbf{r} \bmod q$.

Basic "encryption" transformation. Underlying both the vector and matrix encryption procedure, is the following "encryption" procedure (in quotes, since it does not have a matching decryption procedure). Given the secret key $\mathsf{sk} = (\mathbf{S}, \mathbf{E})$ and a vector $\mathbf{v} \in \mathbb{Z}_q^n$, we draw a noise vector $\mathbf{e} \leftarrow \mathsf{NoiseSamp}(\mathsf{sk}, \mathbf{v})$, then output the "ciphertext"

$$\mathbf{c} := \mathsf{Enc}^*_{\mathsf{sk}}(\mathbf{v}) = \mathbf{S}^{-1}(\mathbf{v} + \mathbf{e}).$$

We remark that the low-order bits of $\mathbf{v}$ are lost in this transformation, due the added noise. Still, the "ciphertext" satisfies the property that $\mathbf{Sc} \approx \mathbf{v}$, up to the low-norm noise vector $\mathbf{e}$.

We provide in Section 4 a detailed proof that the procedure above provides semantic security for $\mathbf{v}$, under the inhomogeneous NTRU hardness assumption.

Vector encryption and decryption. As with Regev encryption [47], to convert the above to real encryption we just need to multiply $\mathbf{v}$ by a large enough scalar β so that $\|\mathbf{e}\|_\infty < \beta$ with high probability. Let b be an upper bound on the l_∞ norm of vectors that can be dealt with (which depends on the application), we assume that $b \ll q$ and set $\beta := \lfloor q/b \rfloor$.

To encrypt a vector $\mathbf{v} \in \mathbb{Z}_b^n$ we just set $\mathbf{c} := \mathsf{VecEnc}_{\mathsf{sk}}(\mathbf{v}) = \mathsf{Enc}^*_{\mathsf{sk}}(\beta \cdot \mathbf{v})$. To decrypt we set $\mathbf{u} := \mathbf{S} \times \mathbf{c} = \beta \cdot \mathbf{v} + \mathbf{e} \pmod q$, then decode each entry of $\mathbf{u}$ to the nearest multiple of β. Namely, we decrypt as

$$\mathbf{v} := \mathsf{VecDec}_{\mathsf{sk}}(\mathbf{c}) = \left\lceil \frac{b \cdot (\mathbf{S} \times \mathbf{c} \bmod q)}{q} \right\rfloor.$$

Matrix encryption and decryption. Matrix encryption is similar, except that instead of just multiplying by a large scalar, we use the GSW technique of redundant encoding using $\mathbf{G}$.

The "native plaintext space" consists of square matrices $\mathbf{M} \in \mathbb{Z}_q^{n\times n}$. To encrypt $\mathbf{M}$ we first compute $\mathbf{M}' = \mathbf{M} \times \mathbf{G} \pmod q$ and let $\mathbf{m}'_j$ be the j'th column of $\mathbf{M}'$ ($j = 1, \ldots, m$). Then we set

$$\mathbf{c}_j := \mathsf{Enc}^*_{\mathsf{sk}}(\mathbf{m}'_j), \text{ and } \mathbf{C} := \mathsf{MatEnc}_{\mathsf{sk}}(\mathbf{M}) = [\mathbf{c}_1|\mathbf{c}_2|\ldots|\mathbf{c}_m].$$

Note that the ciphertext $\mathbf{C}$ has the form $\mathbf{C} = \mathbf{S}^{-1} \times (\mathbf{MG} + \mathbf{E}')$, where $\mathbf{E}'$ is the low-norm matrix consisting of all the noise vectors that were drawn inside of $\mathsf{Enc}^*_{\mathsf{sk}}$. In other words, the property that this ciphertext satisfies is $\mathbf{S} \times \mathbf{C} \approx \mathbf{M} \times \mathbf{G}$, up to the low-norm error matrix $\mathbf{E}'$.

In our application we never need to decrypt matrices, but note that we could compute $\mathbf{U} := \mathbf{S} \times \mathbf{C} = \mathbf{MG} + \mathbf{E}' \pmod q$, and then recover $\mathbf{M}$ from $\mathbf{U}$ (since $\mathbf{E}'$ is low norm and $\mathbf{G}$ is the gadget matrix that has a known trapdoor).

3.3 A Leveled NFA-Homomorphic Scheme

Computing a single product chain. To enable homomorphic computation of a product of k matrices by a vector, $(\prod_{i=k}^{1} \mathbf{M}_i) \times \mathbf{v}$, we choose $k+1$ secret keys as above, $\mathsf{sk}_i = (\mathbf{S}_i, \mathbf{E}_i)$, for $i = 0, 1, \ldots, k$. We then encrypt the vector $\mathbf{v}$ under the first key sk_0, and for $1 \le i \le k$ we use sk_i to encrypt the matrix $\mathbf{M}'_i = \mathbf{M}_i \times \mathbf{S}_{i-1}$. In other words, we prepare the ciphertexts

$$\mathbf{c} = \mathbf{S}_0^{-1} \times (\beta\mathbf{v} + \mathbf{e}) \bmod q,$$

and

$$\mathbf{C}_i = \mathbf{S}_i^{-1} \times (\mathbf{M}_i\mathbf{S}_{i-1}\mathbf{G} + \mathbf{E}'_i) \bmod q, \text{ for } i = 1, \ldots, k,$$

where the noise vectors/matrices are all low-norm. To perform the homomorphic computation, we initialize $\mathbf{c}_0 := \mathbf{c}$, and then repeatedly set

$$\mathbf{c}_i := \mathbf{C}_i \times \mathbf{G}^{-1}(\mathbf{c}_{i-1}) \bmod q,$$

outputting the final vector ciphertext $\mathbf{c}_k$. We now show (by induction) that for every i, the vector ciphertext $\mathbf{c}_i$ is a valid encryption of the plaintext vector $\mathbf{v}_i = (\prod_{j=i}^{1} \mathbf{M}_j) \times \mathbf{v}$ under the key sk_i. This holds by definition for $\mathbf{v}_0 = \mathbf{v}$, so we now assume that it holds for $i \ge 0$ and show for $i+1$. By assumption we have

$$\mathbf{c}_i = \mathbf{S}_i^{-1} \times (\beta\mathbf{v}_i + \mathbf{e}_i),$$

for some low-norm noise vector $\mathbf{e}_i$. Hence we get

$$\begin{aligned}
\mathbf{c}_{i+1} = \mathbf{C}_{i+1} \times \mathbf{G}^{-1}(\mathbf{c}_i) &= \mathbf{S}_{i+1}^{-1} \times (\mathbf{M}_{i+1}\mathbf{S}_i\mathbf{G} + \mathbf{E}'_{i+1}) \times \mathbf{G}^{-1}(\mathbf{c}_i) \\
&= \mathbf{S}_{i+1}^{-1} \times \big(\mathbf{M}_{i+1}\mathbf{S}_i \times \mathbf{c}_i + \mathbf{E}'_{i+1} \times \mathbf{G}^{-1}(\mathbf{c}_i)\big) \\
&= \mathbf{S}_{i+1}^{-1} \times \big(\mathbf{M}_{i+1}\mathbf{S}_i \times \mathbf{S}_i^{-1} \times (\beta\mathbf{v}_i + \mathbf{e}_i) + \mathbf{E}'_{i+1} \times \mathbf{G}^{-1}(\mathbf{c}_i)\big) \\
&= \mathbf{S}_{i+1}^{-1} \times \big(\beta \underbrace{\mathbf{M}_{i+1}\mathbf{v}_i}_{\mathbf{v}_{i+1}} + \underbrace{\mathbf{M}_{i+1}\mathbf{e}_i + \mathbf{E}'_{i+1} \times \mathbf{G}^{-1}(\mathbf{c}_i)}_{\mathbf{e}_{i+1}}\big).
\end{aligned}$$

Since $\mathbf{e}_i, \mathbf{E}'_{i+1}$, and $\mathbf{G}^{-1}(\mathbf{c}_i)$ are all low norm, the noise term $\mathbf{e}_{i+1}$ will be low norm as long as $\mathbf{M}_{i+1}$ is. We conclude that $\mathbf{c}_k = \mathbf{S}_k^{-1}(\beta\mathbf{v}_k + \mathbf{e}_k) \pmod q$, where the noise term is

$$\mathbf{e}_k = (\prod_{j=k}^{1} \mathbf{M}_j)\mathbf{e} \;+\; \sum_{i=2}^{k} (\prod_{j=k}^{i} \mathbf{M}_j)\mathbf{E}'_{i-1}\mathbf{G}^{-1}(\mathbf{c}_{i-2}) \;+\; \mathbf{E}'_k\mathbf{G}^{-1}(\mathbf{c}_{k-1}) \pmod q. \tag{2}$$

Hence as long as all the products $\prod_{j=k}^{i} \mathbf{M}_j$ have low norm, the final noise term $\mathbf{e}_k$ will also have low norm. We will present a detailed analysis on the bounds of the noise terms in relation with NFAs in Section 5.

Encrypting and evaluating an NFA. To be able to evaluate this NFA on strings of up to k symbols, we set the parameters so that $\beta = \lfloor q/b \rfloor$ is sufficiently larger than $\max_{w\in\Sigma^{\le k}} \| \prod_{i=|w|}^{1} \mathbf{M}_{w_i} \|_\infty$, then choose $k+1$ secret keys sk_i for $i = 0, \ldots, k$. We encrypt the initial state vector $\mathbf{v}$ under sk_0, and encrypt each of the matrices $\mathbf{M}_\sigma$ for $\sigma \in \Sigma$ under all the other keys. Namely we set

$$\mathbf{c} = \mathsf{VecEnc}_{\mathsf{sk}_0}(\mathbf{v}), \text{ and } \mathbf{C}_{\sigma,i} = \mathsf{MatEnc}_{\mathsf{sk}_i}(\mathbf{M}_\sigma S_{i-1}) \text{ for } i = 1, \ldots, k.$$

Clearly this method provides semantic security for the NFA, so long as the basic "encryption" transformation from above is semantically secure.

To evaluate the encrypted NFA on a k-symbol string $w_1w_2\ldots w_k$, we apply the chain-product procedure from above to evaluate homomorphically the product $(\prod_{i=k}^{1} \mathbf{M}_{w_i}) \times \mathbf{v}$. Namely we set $\mathbf{c}'_0 = \mathbf{c}$ and then $\mathbf{c}'_i = \mathbf{C}_{w_i,i} \times \mathbf{G}^{-1}(\mathbf{c}'_{i-1})$ for $i = 1, \ldots, k$. At the end of the evaluation, we decrypt the final ciphertext $\mathbf{c}'_k$ to $\mathbf{u} = \mathsf{VecDec}_{\mathsf{sk}_k}(\mathbf{c}'_k)$ and check if the computation is accepting.

Circular Security for Better Efficiency. As usual, we can improve efficiency by assuming circular security of the encryption. Namely, instead of choosing all the secret keys independently, we choose just a single secret key and use it everywhere. This means that we only need the ciphertexts

$$\mathbf{c} = \mathbf{S}^{-1} \times (\beta\mathbf{v} + \mathbf{e}), \text{ and } \mathbf{C}_\sigma = \mathbf{S}^{-1} \times (\mathbf{M}_\sigma\mathbf{SG} + \mathbf{E}_\sigma) \text{ for each } \sigma \in \Sigma.$$

3.4 The Parameters

To determine the parameters that are needed for certain NFA (or a class of NFAs) on k-symbol strings, we first need an upper bound on the size of the plaintext, specifically

$$B_{\mathsf{ptxt}} \ge \max_{w\in\Sigma^{\le k}} \| \prod_{i=|w|}^{1} \mathbf{M}_{w_i} \|_\infty.$$

(See Section 5 for methods of converting regular expressions to NFAs while keeping this bound small.) Once we have the bound B_{ptxt}, we use it on Equation 2 to compute a high probability bound on the expression

$$B^* \ge \|B_{\mathsf{ptxt}} \cdot \mathbf{e} + k \cdot B_{\mathsf{ptxt}} \cdot \mathbf{E} \times \mathbf{G}^{-1}(\mathbf{c})\|,$$

where $\mathbf{e}, \mathbf{E}$ are noise terms that are output by the $\mathsf{NoiseSamp}$ procedure. This value B^* bounds with high probability the size of the noise that we can get when evaluating the NFA, and so we need to choose $q > B^* \cdot B_{\mathsf{ptxt}}$ (since our plaintext can be as large as B_{ptxt}).

At the same time, we need to set n large enough relative to q to ensure the required security level (say $q < 2^{n/\lambda}$), and $m > O(n \log q)$ (since we rely on the leftover hash lemma). As usual with lattice-based systems, there is a weak circular dependence between these constraints, but it is not hard to find values that satisfy them all.

4 Security Analysis

Below we define (two variants of) the inhomogeneous NTRU problem, one over a ring and one over integer matrices. We describe some properties of this problem, and show that hardness of the matrix variant implies the security of our encryption scheme.

4.1 Inhomogeneous NTRU

We begin with the ring variant of our hardness assumption. Fix a ring R, a modulus q, and an error distribution χ over R, producing with overwhelming probability elements with norm $\ll q$ and $-\chi = \chi$. Denoting $\ell = \lceil \log q \rceil$, the iNTRU distribution with these parameters is defined as follows:

$$\mathsf{iNTRU} = \left\{ \begin{array}{r} \text{draw } s \leftarrow R/qR, \text{ and } e_i \leftarrow \chi, \text{ for } i = 0, \ldots, \ell, \\ \text{set } a_0 := e_0/s \bmod q, \\ \text{and } a_i := (2^{i-1} - e_i)/s \bmod q \text{ for } i = 1, \ldots, \ell, \\ \text{output } (a_0, \ldots, a_{\ell-1}) \end{array} \right\}. \tag{3}$$

The inhomogeneous NTRU problem is to distinguish between this distribution and the uniform distribution over $(R/qR)^\ell$.

In the matrix variant of this assumption, the ring elements s, e_i are replaced by n-by-n integer matrices, and the a_i's are similarly replaced with matrices $\mathbf{A}_0 := -\mathbf{S}^{-1} \times \mathbf{E}_0$, $\mathbf{A}_i := \mathbf{S}^{-1} \times (2^i\mathbf{I} - \mathbf{E}_i)$. In matrix notation, let $m' = n(\ell + 1)$ and $\mathbf{G}'$ be the gadget matrix[11] $\mathbf{G}' = [\mathbf{0}|\mathbf{I}|2\mathbf{I}|4\mathbf{I}|\ldots|2^{\ell-1}\mathbf{I}] \in \mathbb{Z}^{n\times m'}$, and let χ be a distribution over $\mathbb{Z}$, producing with overwhelming probability integers of

[11] We use a slightly larger gadget matrix than usual, with an extra first block. The reason will become clear when we prove Lemma 4.1 below.

magnitude $\ll q$. The matrix-iNTRU distribution (MiNTRU) with these parameters is defined as follows:

$$\mathsf{MiNTRU} = \left\{ \begin{array}{l} \text{draw } \mathbf{S} \leftarrow \mathbb{Z}_q^{n\times n}, \text{ and } \mathbf{E}' \leftarrow \chi^{n\times m'}, \\ \quad \text{output } \mathbf{A}' := \mathbf{S}^{-1} \times (\mathbf{G}' - \mathbf{E}') \bmod q \end{array} \right\}. \tag{4}$$

As before, the hardness assumption says that MiNTRU is pseudorandom, namely that the matrix $\mathbf{A}'$ is indistinguishable from a matrix uniform in $\mathbb{Z}_q^{n\times m'}$.

4.1.1 Small-Secret Inhomogeneous NTRU

Similarly to LWE, here too we can prove that the inhomogeneous NTRU problem remains hard even when the secret is chosen from the error distribution. We lose a little on parameters in the conversion, specifically the extra block at the beginning of $\mathbf{G}'$. With the parameters n, m', q, χ as above, let $m = n\lceil \log q \rceil = m' - n$, and $\mathbf{G} = [\mathbf{I}|2\mathbf{I}|4\mathbf{I}|\ldots|2^{\ell-1}\mathbf{I}] \in \mathbb{Z}^{n\times m}$. The matrix-iNTRU distribution with small secret (MiNTRU^s) is as follows:

$$\mathsf{MiNTRU}^s = \left\{ \begin{array}{l} \text{draw } \mathbf{S} \leftarrow \chi^{n\times n}, \text{ and } \mathbf{E} \leftarrow \chi^{n\times m}, \\ \quad \text{output } \mathbf{A} := \mathbf{S}^{-1} \times (\mathbf{G} - \mathbf{E}) \bmod q \end{array} \right\}. \tag{5}$$

Lemma 4.1. *For the parameters n, m, m', q, χ as above, if MiNTRU is pseudorandom in $\mathbb{Z}_q^{n\times m'}$, then MiNTRU^s is pseudorandom in $\mathbb{Z}_q^{n\times m}$.*

Proof. We show that if we could distinguish MiNTRU^s from uniformly random n-by-m matrices over $\mathbb{Z}_q$ then we could also distinguish MiNTRU from uniformly random n-by-m' matrices over $\mathbb{Z}_q$. Given a MiNTRU instance that we want to distinguish, $\mathbf{A}' = [\mathbf{A}'_0|\mathbf{A}'_1|\ldots|\mathbf{A}'_\ell]$ (with $\mathbf{A}'_i \in \mathbb{Z}_q^{n\times n}$), we set

$$\mathbf{A}_i = \mathbf{A}_0'^{-1} \times \mathbf{A}'_i \bmod q, \text{ for } i = 1, \ldots, \ell,$$

(aborting if $\mathbf{A}'_0$ is not invertible), then run the MiNTRU^s distinguisher on $\mathbf{A} = [\mathbf{A}_1|\mathbf{A}_2|\ldots|\mathbf{A}_\ell]$. Observe that if $\mathbf{A}'$ is uniformly random then so is $\mathbf{A}$, and if $\mathbf{A}'$ is chosen from the MiNTRU distribution then

$$\mathbf{A}_i = \mathbf{A}_0'^{-1} \times \mathbf{A}'_i = -\mathbf{E}_0'^{-1} \times \mathbf{S} \times \mathbf{S}^{-1} \times (2^{i-1}\mathbf{I} - \mathbf{E}'_i) = -\mathbf{E}_0'^{-1} \times (2^{i-1}\mathbf{I} - \mathbf{E}'_i),$$

for $i = 1, \ldots, \ell$, and hence $\mathbf{A}$ follows the MiNTRU^s distribution as needed. ☐

4.2 Security Reduction

We next show that pseudorandomness of MiNTRU^s (or equivalently MiNTRU) with some error distribution χ, implies the semantic security of our scheme with a related error distribution (but not quite the same). Specifically, let n, m, q, χ be the parameters of the MiNTRU^s distribution above. For a fixed pair of matrices $\mathbf{E}, \mathbf{Y} \in \mathbb{Z}_q^{n\times m}$, consider the distribution

$$\psi[\mathbf{E}, \mathbf{Y}] = \{\mathbf{R} \leftarrow \mathbf{G}^{-1}(\mathbf{Y}), \text{ output } \mathbf{E} \times \mathbf{R} \bmod q\}.$$

In the provable version of our scheme, the secret key includes the square invertible matrix $\mathbf{S} \leftarrow \chi^{n\times n}$, and in addition a fixed error matrix $\mathbf{E} \leftarrow \chi^{n\times m}$, and we use the error distribution $\psi[\mathbf{E}, \mathbf{M}\times\mathbf{G}]$ when encrypting a matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times n}$. Namely we draw a sample $\mathbf{R} \leftarrow \mathbf{G}^{-1}(\mathbf{MG}) \in \mathbb{Z}_q^{m\times m}$, then output the ciphertext $\mathbf{C} := \mathbf{S}^{-1} \times (\mathbf{MG} - \mathbf{ER}) \bmod q$. Note that given a MiNTRU^s sample $\mathbf{S}^{-1} \times (\mathbf{G} - \mathbf{E})$, one can efficiently generate samples of the form $\mathbf{S}^{-1} \times (\mathbf{M}_i\mathbf{G} - \mathbf{ER})$. This means Proposition 4.2 is a reduction from CPA security to distinguishing a single MiNTRU^s sample.

Proposition 4.2. *If MiNTRU^s is pseudorandom, then our encryption scheme using the error distribution $\psi[\mathbf{E}, \mathbf{M} \times \mathbf{G}]$ is semantically secure.*

Proof. We use the "real-or-random" formulation of semantic security for secret-key encryption [9]. Namely, we have a challenger that chooses a secret key $\mathsf{sk} = (\mathbf{S}, \mathbf{E})$, where $\mathbf{S} \leftarrow \chi^{n\times n}, \mathbf{E} \leftarrow \chi^{n\times m}$, and a bit $\sigma \leftarrow \{0, 1\}$, then the adversary repeatedly chooses messages $\mathbf{M}_i \in \mathbb{Z}_q^{n\times n}$ for $i = 1, \ldots, k$ and sends them to the challenger, who replies either with uniformly random matrices $\mathbf{C}_i \in \mathbb{Z}_q^{n\times m}$ if $\sigma = 0$, or with ciphertexts $\mathbf{C}_i := \mathsf{MatEnc}_{\mathsf{sk}}(\mathbf{M}_i) = \mathbf{S}^{-1} \times (\mathbf{M}_i\mathbf{G} + \mathbf{E}_i)$ if $\sigma = 1$, where $\mathbf{E}_i \leftarrow \psi[\mathbf{E}, \mathbf{M}_i\mathbf{G}]$, for $i = 1, \ldots, k$. The adversary eventually outputs a guess σ' for σ, and is considered successful if $\sigma' = \sigma$ with probability significantly larger than 1/2.

We show that an adversary Adv with a noticeable advantage ϵ can be transformed into a distinguisher between MiNTRU^s and the uniform distribution over $\mathbb{Z}_q^{n\times m}$, with an advantage close to ϵ. The distinguisher D receives as input $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ that is either an instance of MiNTRU^s or a uniformly random matrix, and it interacts with the adversary Adv as follows:

When receiving a matrix $\mathbf{M}_i$ from Adv, the distinguisher D draws a sample $\mathbf{R}_i \leftarrow \mathbf{G}^{-1}(\mathbf{M}_i\mathbf{G})$, and replies with the "ciphertext" $\mathbf{C}_i := \mathbf{AR}_i \bmod q$. When Adv eventually outputs a guess σ', the distinguisher D outputs the same guess. We next show that the distinguishing advantage of D is very close to ϵ.

If $\mathbf{A}$ is a uniformly random matrix in $\mathbb{Z}_q^{n\times m}$ then, by the leftover hash lemma, each $\mathbf{C}_i = \mathbf{A} \times \mathbf{G}^{-1}(\mathsf{something}) \bmod q$ is statistically close to uniformly random matrices in $\mathbb{Z}_q^{n\times m}$ and independent of $\mathbf{A}$. On the other hand, if $\mathbf{A} = \mathbf{S}^{-1} \times (\mathbf{G} - \mathbf{E})$ is an instance of MiNTRU^s, then we have

$$\begin{aligned}
\mathbf{C}_i = \mathbf{A} \times \mathbf{G}^{-1}(\mathbf{M}_i\mathbf{G}) &= \mathbf{S}^{-1} \times \big(\mathbf{G} \times \mathbf{G}^{-1}(\mathbf{M}_i\mathbf{G}) - \mathbf{E} \times \mathbf{G}^{-1}(\mathbf{M}_i\mathbf{G})\big) \\
&= \mathbf{S}^{-1} \times \big(\mathbf{M}_i\mathbf{G} - \mathbf{E} \times \mathbf{G}^{-1}(\mathbf{M}_i\mathbf{G})\big),
\end{aligned}$$

which is identical to the distribution produced by our encryption procedure. ☐

4.3 Hardness of MiNTRU from LWE with a Trapdoor

Here we prove the reduction alluded to in Section 1.2. We define a trapdoor oracle for an arbitrary matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times m}$ as an oracle which takes as input $\mathbf{B}$, a vector $\mathbf{v} \in \mathbb{Z}_q^n$, and outputs a discrete Gaussian integer vector $\mathbf{x} \in \mathbb{Z}^m$ conditioned on $\mathbf{Bx} \bmod q = \mathbf{v}$. Repeated calls to the oracle are assumed to use independent random coins. Further, we assume the oracle's distribution samples above the smoothing parameter of

$$\Lambda_q^\perp(\mathbf{B}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{Bx} = \mathbf{0} \bmod q\}$$

for a *uniformly random* $\mathbf{B}$, for some negligible function $\epsilon(n)$. In general, the smoothing parameter of $\Lambda_q^\perp(\mathbf{B})$ is just above the smoothing parameter of $\mathbb{Z}^m$, for some negligible $\epsilon(n)$, when $m > n \log q$, [43, Lemma 2.4].

Let n-secret LWE define the distribution

$$\{(\mathbf{A}, \mathbf{B} = \mathbf{SA} + \mathbf{E}) : \mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{S} \leftarrow \mathbb{Z}_q^{n\times n}, \mathbf{E} \leftarrow \chi^{n\times m}\}$$

for some distribution χ. Next, we show the pseudorandomness of MiNTRU follows from the n-secret LWE distribution with a trapdoor oracle for $\mathbf{B}$. Let $\mathbf{G} \in \mathbb{Z}_q^{n\times m'}$ be any formulation of the gadget matrix. ($\mathbf{G} = [\mathbf{0}|\mathbf{I}|2\mathbf{I}|\cdots|2^{\log q-1}\mathbf{I}] \in \mathbb{Z}_q^{n\times n(\log q+1)}$ in the MiNTRU definition.)

Proposition 4.3. *Let $n \in \mathbb{N}$, $q < 2^{\mathrm{poly}(n)}$, χ be a distribution over $\mathbb{Z}_q$, $m \ge n \log q$, and m' be the number of columns in the $\mathbf{G}$-matrix. Further, let $q = \omega(\sqrt{m})$. Then, the pseudorandomness of MiNTRU with error distribution $\chi^{n\times m} \cdot \mathbf{B}^{-1}(\mathbf{G})$ follows from the pseudorandomness of n-secret LWE with a trapdoor oracle for $\mathbf{B}$.*

Proof. We show a reduction from the n-secret LWE with a trapdoor oracle for $\mathbf{B}$ to MiNTRU with error distribution $\chi^{n\times m} \cdot \mathbf{B}^{-1}(\mathbf{G})$. Given as input a pair of matrices $(\mathbf{A}, \mathbf{B})$, we call m' times the trapdoor oracle for $\mathbf{B}$ to get $\mathbf{X} \leftarrow \mathbf{B}^{-1}(\mathbf{G})$. Then the reduction outputs $\mathbf{A} \times \mathbf{X} \bmod q$. Notice when $(\mathbf{A}, \mathbf{B})$ is generated uniformly and independently, then $\mathbf{AX} \bmod q$ is negligibly close to uniformly random by leftover hash lemma, along with Lemmas 2.3 and 2.4. Conversely, we have $\mathbf{S}^{-1} \in \mathbb{Z}_q^{n\times n}$ exists with high probability and $\mathbf{A} = \mathbf{S}^{-1} \times (\mathbf{B} - \mathbf{E}) \bmod q$ when $(\mathbf{A}, \mathbf{B})$ is sampled from the n-secret LWE distribution. Therefore,

$$\mathbf{A} \times \mathbf{B}^{-1}(\mathbf{G}) = \mathbf{S}^{-1} \times (\mathbf{G} - \mathbf{EB}^{-1}(\mathbf{G})) = \mathbf{S}^{-1} \times (\mathbf{G} - \mathbf{E}') \bmod q.$$

So $\mathbf{AX} \bmod q$ is an instance of MiNTRU with the desired error distribution. ☐

Remark. There is an identical reduction from n-secret LWE with a trapdoor for $\mathbf{B}$ with small secrets to MiNTRU^s.

5 Converting Regular Expressions to Automata

In real world applications, regular languages or finite automata are often represented by regular expressions, which have a very compact form and are convenient to store. So it is important for our scheme to be useful when NFAs are specified using regular expressions. In this section we present an efficient method to convert regular expressions to NFAs of relatively small sizes, and we discuss how to find a suitable NFA to bound the noise growth. We assume the reader has some familiarity with regular languages, regular expressions, and finite automata. See Appendix A for basic notation and definitions.

Partial derivatives and NFAs. Let Σ be a finite alphabet, and RE be the set of all regular expressions over Σ. We consider the basic operations such as union ("+"), concatenation ("·"), and Kleene star ("∗") on regular expressions. For any regular expression e, the *language* of e is denoted by $\mathcal{L}(e)$. To convert a regular expression to an NFA, we start with Antimirov's partial derivative construction [4], which is an elegant extension of Brzozowski's derivative construction [15] to NFAs. For any symbol $a \in \Sigma$, the *partial derivative* of e w.r.t. a, denoted as $\partial_a(e)$, is a set of regular expressions defined inductively as

$$\partial_a(\epsilon) = \emptyset, \qquad \partial_a(e_0 + e_1) = \partial_a(e_0) \cup \partial_a(e_1), \qquad \partial_a(e^*) = \partial_a(e)e^*$$

$$\partial_a(a_i) = \begin{cases} \{\epsilon\} & \text{if } a_i = a \\ \emptyset & \text{otherwise} \end{cases} \qquad \partial_a(e_0 \cdot e_1) = \begin{cases} \partial_a(e_0)e_1 \cup \partial_a(e_1) & \text{if } \epsilon \in \mathcal{L}(e_0) \\ \partial_a(e_0)e_1 & \text{otherwise} \end{cases}$$

where e, e_0, e_1 range over RE. The partial derivative of e w.r.t. any string is $\partial_\epsilon(e) = \{e\}$ and $\partial_{ua}(e) = \bigcup\{\partial_a(f) \mid f \in \partial_u(e)\}$ where $u \in \Sigma^*$ and $a \in \Sigma$. A regular expression e' is a *partial derivative term* of e if e' is an element of $\partial_w(e)$ for some $w \in \Sigma^*$, and $\partial(e)$ is the set of all partial derivative terms of e.

Definition 1 (Partial derivative NFA)**.** For any regular expression e, the *partial derivative NFA* of e is $\mathcal{M}_{\mathcal{PD}}(e) = (Q, \Sigma, \delta, Q_I, Q_F)$, where $Q = \partial(e)$, $Q_I = \{e\}$, $Q_F = \{e' \in \partial(e) \mid \epsilon \in \mathcal{L}(e')\}$, and for any $e' \in Q$ and $a \in \Sigma$, $\delta(e', a) = \partial_a(e')$.

Remark. It was shown in [4] that $\partial(e)$ is a finite set (with respect to syntactic equality on regular expressions). In fact, $|\partial(e)| \le r + 1$ where r is the number of occurrences of alphabet symbols in e.

The language of e satisfies $\mathcal{L}(e) = \bigcup_{a\in\Sigma} a \cdot \partial_a(e)$. It follows that the language accepted by $\mathcal{M}_{\mathcal{PD}}(e)$ is exactly $\mathcal{L}(e)$.

Ambiguity measure. As will be shown later, when evaluating an encrypted NFA, the noise growth is closely related to the amount of nondeterministic choices of the NFA. Here we describe some notions that characterize this quantity. Let $\mathcal{M} = (Q, \Sigma, \delta, Q_I, Q_F)$ be an NFA. For any string $w = w_1 \cdots w_k$ where $w_1, \ldots, w_k \in \Sigma$, a *path of w from state s to state t* is a finite sequence of states $s = s_{i_0}, s_{i_1}, \ldots, s_{i_k} = t$ such that $s_{i_j} \in \delta(s_{i_{j-1}}, w_j)$ for all $1 \le j \le k$. A path is accepting if $s \in Q_I$ and $t \in Q_F$. The *degree of ambiguity of $\mathcal{M}$*, denoted as $\mathrm{da}(\mathcal{M}, k)$, is the maximal number of accepting paths for a string of length k. If $\mathrm{da}(\mathcal{M}, k) \le 1$ for all $k > 0$, then we say $\mathcal{M}$ is *unambiguous*.[12] We say that $\mathcal{M}$ is *finitely ambiguous* if $\sup\{\mathrm{da}(\mathcal{M}, k) \mid k \ge 0\} < \infty$, and $\mathcal{M}$ is infinitely ambiguous otherwise. Clearly $\mathrm{da}(\mathcal{M}, k) \le |Q|^{k+1}$ for any NFA. To upper bound the quantity $\mathrm{da}(\mathcal{M}, k)$ using a function of k, we can define the *degree of growth of ambiguity of $\mathcal{M}$*, denoted as $\deg(\mathcal{M})$, to be the minimal *degree* of a polynomial $h(\cdot)$ such that $\mathrm{da}(\mathcal{M}, k) \le h(k)$ for all $k \ge 0$. If no such polynomial exists, we simply set $\deg(\mathcal{M}) = \infty$. Note that $\mathcal{M}$ is finitely ambiguous if and only if $\deg(\mathcal{M}) = 0$. It was shown in [54] that $\deg(\mathcal{M})$ can be computed in time $O(r^6|\Sigma|)$ for any NFA $\mathcal{M}$ with r states.

On optimizing NFA. For our application of evaluating encrypted NFA, an optimal NFA should be such that its encryption can be correctly evaluated on as many strings as possible. Concretely, we want to find an NFA such that the noise term at the end of evaluation is small enough for a successful decryption. Recall that (n, q) is the lattice parameter in our scheme, b is the maximum l_∞ norm on plaintext vectors, and χ is an error distribution from which we sample noise terms. As we assume the first state will be the only initial state in all our NFAs, we can encrypt the initial state vector with no noise. As a result, we obtain the following bounds on the noise due to homomorphic evaluation of NFAs, which can be bounded using the ambiguity measures of $\mathcal{M}$.

Proposition 5.1. *For any $n \ge 1$, if $\mathcal{M}$ is an NFA with $r \le n$ states, and w a string of length k, the noise vector $\mathbf{e}^{(k)}$ at the end of homomorphic evaluation of encrypted $\mathcal{M}$ on w satisfies the following bounds:*

- *If $\mathcal{M}$ is unambiguous, then $\|\mathbf{e}^{(k)}\|_\infty \le bnk\chi \log_b q$.*
- *If $\mathcal{M}$ is finitely ambiguous, then $\|\mathbf{e}^{(k)}\|_\infty \le bnrk\chi \log_b q$.*
- *If $\mathcal{M}$ is infinitely ambiguous, then $\|\mathbf{e}^{(k)}\|_\infty \le bnk^{\deg(\mathcal{M})+1}\chi \log_b q$.*

[12] Notice that a DFA $\mathcal{M}$ has $\mathrm{da}(\mathcal{M}, k) \le 1$ for all $k \ge 0$, but the converse is not necessarily true. An NFA can have multiple nondeterministic choices at every state but still satisfies $\mathrm{da}(\mathcal{M}, k) \le 1$, in such cases at most one of these choices could lead to a final state.

Notice that both the number of states and the degree of ambiguity contribute to the bound on the noise growth. To find a small noise growth for the general case of processing an arbitrary long input string, we can try to solve the following optimization problem on NFA minimization with bounded ambiguity.

Definition 2 (NFA Minimization with Bounded Ambiguity Problem)**.** For a given NFA of r states and a function $B : \mathbb{N} \to \mathbb{N}$, find an equivalent NFA $\mathcal{M}$ with a minimal number of states such that $\mathrm{da}(\mathcal{M}, k) \le B(k)$ for all $k \ge 1$.

A closely related problem is to find a minimal NFA $\mathcal{M}$ with a given bound on $\deg(\mathcal{M})$. Conversely, we can consider a similar minimization problem of finding an NFA $\mathcal{M}$ with minimal $\deg(\mathcal{M})$ when given a regular expression and a bound on the number of states. These problems seem to be hard in general as evidenced by several exponential separation results in automata theory, and we briefly mention a few. It was shown in [37] that, for each $r > 0$, there exists an NFA of r states such that the minimal equivalent NFA $\mathcal{M}'$ of bounded $\deg(\mathcal{M}')$ have $2^r - 1$ states.[13] With a more strict bound on the ambiguity, it was known [32] that there exist NFAs of r states such that the equivalent finitely ambiguous NFAs have at least $2^{\Omega(r^{1/3})}$ states. A more tractable problem of finding a minimal unambiguous NFA is NP-complete [33, 10].

On the other hand, unambiguous NFAs can have much smaller size than equivalent DFAs. A well-known example is the language $L_r = (0 + 1)^*0(0 + 1)^{r-2}$ for any $r \ge 2$: its partial derivative NFA has r states and is unambiguous, but its minimal equivalent DFA requires 2^{r-1} states [41]. The exponential upper bound 2^r can actually be met: it was shown in [38] that there exists a series $\{\mathcal{M}_r\}_{r\ge1}$ of unambiguous NFAs such that $\mathcal{M}_r$ has r states but the minimal equivalent DFA of $\mathcal{M}_r$ has 2^r states. Notice that, if the size of the given regular expression is small, the bound on the size of the noise is dominated by the degree of ambiguity, which is same for unambiguous NFAs and DFAs. So we can exploit the fact that our scheme supports homomorphic encryption of NFAs and try to find a small unambiguous NFA, which can be much more efficient than encrypting DFAs.

Some particular useful classes of regular languages are the pattern matching languages L such that $L = \Sigma^*K\Sigma^*$, $L = K\Sigma^*$, or $L = \Sigma^*K$ where K is a finite set of strings. One can check using the criterion in [54] that the partial derivative NFA for such a language is unambiguous, but its minimal equivalent DFA may have exponentially many states. Even if K can be specified using a DFA of m states, the minimal equivalent DFA of L may still have $2^{m-2} + 1$ states. As our scheme supports encryption of NFAs, pattern matching on encrypted patterns can be much more efficient than previous approaches via DFAs.

6 Implementation and Performance

This section describes a proof of concept implementation of our scheme[14] and compares its performance with the HAO15 matrix-FHE scheme [29] when applied to homomorphic evaluation of encrypted NFAs.

Implementation. We implemented our scheme in C++ using the NTL library (version 10.5.0) for a power of two modulus, q, and we performed experiments on an Intel i7-2600 3.4 GHz CPU.

[13] Note that $\deg(\mathcal{M}')$ is bounded if and only if $\mathrm{da}(\mathcal{M}', k)$ is at most a polynomial in k for all $k > 0$.

[14] The source code of our proof-of-concept implementation can be accessed at `https://www.dropbox.com/s/10g2nocx3pmyu4t/henfa.zip`

Input Length (4k)	NFA Enc. Time	Matching	Enc. NFA	RAM used
256 bit S.L.	16.35 sec	1.53 sec	66Mb	172Mb
512 bit S.L.	16.66 sec	3.34 sec	66Mb	172Mb
1024 bit S.L.	16.53 sec	6.63 sec	66Mb	172Mb
16384 bit S.L.	16.76 sec	98.97 sec	66Mb	172Mb
65536 bit S.L.	16.42 sec	394.47 sec	66Mb	172Mb

Table 1: Running times for each function along with memory for a 1024-state NFA accepting the language $(0+1)^*0(0+1)^{r-1}$ for $r = 11$. "NFA Enc. Time" is the time to encrypt the NFA, "Matching" is the time to evaluate an encrypted NFA on an input of k symbols, "Enc. NFA" is the memory storage for the encrypted NFA, and the last column measures the total RAM used during encryption, evaluation, and decryption. Total RAM usage was measured with the "sys/resource.h" library in unix.

The implementation is naive in that it only uses NTL's native functionality with no further optimizations. It can be done in a few hundred lines of code and a few days' programming effort. There are many opportunities for optimization since the code was written for simplicity and not efficiency. Despite this, we noticed exceptionally fast evaluation times as listed in Table 1.

In our experiments, we set lattice parameters to $n = 1024$, $q = 2^{42}$, and $\alpha = \sqrt{2n}/q$. We kept the modulus both as a power of two and as a power of the maximum l_∞ norm b on plaintext vectors in order to take advantage of bit-shifting instead of multiplications and divisions modulo q. The noise matrices $\mathbf{E}_i \leftarrow \chi_q^{n\times m}$ and the secret keys $\mathbf{S} \leftarrow \chi_q^{n\times n}$ were chosen as uniformly random binary matrices with the latter being invertible modulo q.

Notice that MiNTRU^s can be cryptanalyzed by NTRU attacks like dimension reduction [40] , the hybrid attack [31], and the overstretched attack [34] for key recovery. We use the methods in [34] to estimate the concrete security of our scheme, whish shows that our scheme achieves 51 bits of security with these parameters. More detailed analysis can be found in Section D.

We conducted tests on r-state partial derivative NFAs accepting the pattern-matching languages $(0+1)^*0(0+1)^{r-1}$ with finite ambiguity, for some r smaller than the lattice dimension n. Notice that the equivalent minimal DFA's have 2^{r-1} states. In the experiments, we pad the transition matrices to n-dimensional matrices by adding transitions from nonreachable states to final states to increase ambiguity, and hence we effectively obtain n-state NFAs. The strings scanned were randomly generated. At the end of each scan, our code checked for any decryption errors. We observed no decryption errors nor noise overflow. The experiment results for $r = 11$ are listed in Table 1, where time was measured using C++'s "time.h" library.

Consider the worst case where the NFA has infinite ambiguity, but bounded degree of growth of ambiguity. Then the final noise term $\mathbf{e}^{(k)}$ has norm $\|\mathbf{e}^{(k)}\|_\infty \leq bnk^{\deg(\mathcal{M})+1}\chi \log_b q$ as discussed in the previous section. By setting the modulus just above the error growth, we see that the bit length of the modulus is linear in $\deg(\mathcal{M}) + 1$. Now as we view total memory for the encrypted NFA, $n^2|\Sigma|\log_2(q)\log_b(q)$ bits, we see that efficiency is quadratic in NFA's number of states and quadratic in the degree of growth of ambiguity (though we have some control over $\log_b(q)$ by choosing a large base b). This gives us an exact relation between the number of states, the NFA's ambiguity, and performance.

Lattice parameters	$n = 1024, q = 2^{42}$		$n = 1024, q = 2^{33}$		$n = 4096, q = 2^{73}$	
	Ours	HAO15	Ours	HAO15	Ours	HAO15
Unambiguous	564918	141229	1404	351	8.724e13	2.181e13
Finitely ambiguous	551	137	1	-	2.130e10	5.325e9
Infinitely ambiguous	82	65	-	-	44352	35202
Bit security	51	98	108	134	103	231

Table 2: Maximal lengths of strings can be scanned on any n-state NFA in both schemes without decryption error. In some cases the maximal length is less than 1, which is denoted by "-". In all cases, the noise parameter is set to $\alpha = \sqrt{2n}/q$. The bit security of our scheme is estimated using the methods of [34] with more detailed analysis in Section D. For HAO15, we use the on-line LWE bit security estimator `https://bitbucket.org/malb/lwe-estimator` to estimate the time for a uSVP attack as in [1].

Performance improvement over HAO15. Now we compare the performance of our scheme with the HAO15 matrix-FHE scheme for homomorphic evaluation of encrypted NFAs. Let $\mathcal{M}$ be an NFA of $r \leq n$ states, where n is the lattice dimension, and let k be the length of the string to be scanned on $\mathcal{M}$. For the running time, the computational complexity of k homomorphic matrix multiplications in the HAO15 scheme, assuming naive matrix-vector multiplication of complexity $O(n^2)$, is $O(k(r+n)^2 \log q)$. On the other hand, the complexity of our homomorphic evaluation procedure is $O(kn^2 \log q)$. So using the same parameter and matrix multiplication algorithm, we expect our scheme runs three times faster than an implementation of the HAO15 scheme.

To compare the capabilities of homomorphic NFA evaluation, we apply the NFA ambiguity analysis technique as in Proposition 5.1 to the HAO15 scheme. We can rewrite Equation 1 to obtain the following bound on the l_∞ norm of the final noise vector $\mathbf{e}_k$:

$$\|\mathbf{e}_k\|_\infty \leq \chi(n+r)\log q + \chi(n+r)\log q \sum_{l=2}^{k} \mathrm{da}(\mathcal{M}, l) + \chi \mathrm{da}(\mathcal{M}, k), \tag{6}$$

which must be bounded away from $q/4$ for successful decryption of the final ciphertext vector.

Using Proposition 5.1 and the bound in Equation 6, one can determine each scheme's capability of homomorphic NFA evaluation. For concrete results, we consider three cases of the ambiguity of $\mathcal{M}$:

1. $\mathcal{M}$ is unambiguous, so $\mathrm{da}(\mathcal{M}, l) \leq 1$;
2. $\mathcal{M}$ is finitely ambiguous, so $\mathrm{da}(\mathcal{M}, l) \leq r$; and
3. $\mathcal{M}$ is infinitely ambiguous and its degree of growth of ambiguity is $\deg(\mathcal{M}) = 2$, so $\mathrm{da}(\mathcal{M}, l) \leq l^2$.

Furthermore, we consider three sets of lattice parameters for various bit security estimates and maximal sizes r for $\mathcal{M}$. We list in Table 2 the maximal lengths of strings can be scanned without decryption error using both schemes on any n-state NFA. The results imply that there exist tradeoffs among the bit security level, the maximal length of strings can be scanned, and the running time.

Potential Optimizations. One potential optimization is parallelization through the unused states. Say we must evaluate a long string (10000 bits) but only use a 100 state NFA. Then, we can evaluate ten such NFAs in parallel by setting the transition matrix for symbol $a \in \Sigma$ as the block diagonal matrix with the blocks as the smaller transition matrices in the small parameter setting. The total number of states must stay above a few hundred for this corresponds to the lattice dimension of the underlying lattice problem.

Let $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}^t$ for $\mathbf{g}^t = (1, b, \cdots, b^{\log_b(q)-1})$ as in [43]. We expect to see smaller noise growth via a randomized bit decomposition for the decomposition of the encrypted state vector, as used in [3]. This can be done with a simple tweak to Babai's nearest plane algorithm [5] on the G-matrix's null lattice $\Lambda_q^\perp(\mathbf{G}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{G}\mathbf{x} = \mathbf{0} \bmod q\}$ and its cosets.

Acknowledgment

We thank Pierre-Alain Fouque and Paul Kirchner for their helpful discussions as well as the anonymous reviewers for their helpful feedback and suggestions.

References

[1] M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. Estimate all the {LWE, NTRU} schemes! *IACR Cryptology ePrint Archive*, 2018:331, 2018.

[2] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In *USENIX Security Symposium*, pages 327–343. USENIX Association, 2016.

[3] J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology - CRYPTO 2014, Part I*, pages 297–314. Springer, 2014.

[4] V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996.

[5] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[6] S. Bai and S. D. Galbraith. An improved compression technique for signatures based on learning with errors. In *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, pages 28–47, 2014.

[7] W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(1):625–635, 1993.

[8] A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, pages 10–24. SIAM, 2016.

[9] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. In *Proceedings of 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*, pages 394–403. IEEE Press, 1997.

[10] H. Björklund and W. Martens. The tractability frontier for NFA minimization. *J. Comput. Syst. Sci.*, 78(1):198–210, 2012.

[11] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):13, 2014.

[13] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 575–584. ACM, 2013.

[14] Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In M. Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14*, pages 1–12. ACM, 2014.

[15] J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[16] Y. Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Paris 7, 2013.

[17] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song. Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.

[18] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 3–33. Springer, 2016.

[19] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 377–408. Springer, 2017.

[20] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

[21] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[22] N. Gama, M. Izabachène, P. Q. Nguyen, and X. Xie. Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 528–558. Springer, 2016.

[23] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st ACM Symposium on Theory of Computing – STOC 2009*, pages 169–178. ACM, 2009.

[24] C. Gentry, S. Gorbunov, and S. Halevi. Graph-induced multilinear maps from lattices. In Y. Dodis and J. B. Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 498–527. Springer, 2015.

[25] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In C. Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 197–206. ACM, 2008.

[26] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013, Part I*, pages 75–92. Springer, 2013.

[27] S. Halevi and V. Shoup. Faster homomorphic linear transformations in HElib. *IACR Cryptology ePrint Archive*, 2018:244, 2018.

[28] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[29] R. Hiromasa, M. Abe, and T. Okamoto. Packing messages and optimizing bootstrapping in GSW-FHE. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, pages 699–715, 2015.

[30] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In J. Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[31] N. Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 150–169. Springer, 2007.

[32] J. Hromkovic and G. Schnitger. Ambiguity and communication. *Theory Comput. Syst.*, 48(3):517–534, 2011.

[33] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.

[34] P. Kirchner and P. Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 3–26, 2017.

[35] T. Laarhoven. *Search Problems in Cryptography*. PhD thesis, Eindhoven University of Technology, 2015.

[36] C. Lee and A. Wallet. Lattice analysis on mintru problem. *IACR Cryptology ePrint Archive*, 2020:230, 2020.

[37] H. Leung. Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. *SIAM J. Comput.*, 27(4):1073–1082, 1998.

[38] H. Leung. Descriptional complexity of NFA of different ambiguity. *Int. J. Found. Comput. Sci.*, 16(5):975–984, 2005.

[39] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[40] A. May and J. H. Silverman. Dimension reduction methods for convolution modular lattices. In Silverman [50], pages 110–125.

[41] A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, October 13-15, 1971*, pages 188–191, 1971.

[42] D. Micciancio. Improving lattice based cryptosystems using the hermite normal form. In Silverman [50], pages 126–145.

[43] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, 2012.

[44] D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 372–381. IEEE Computer Society, 2004.

[45] G. Pataki and M. Tural. On sublattice determinants in reduced bases. *arXiv preprint arXiv:0804.4014*, 2008.

[46] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 145–166. Springer, 2006.

[47] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[48] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[49] C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[50] J. H. Silverman, editor. *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*, volume 2146 of *Lecture Notes in Computer Science*. Springer, 2001.

[51] N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014. Early verion at `http://eprint.iacr.org/2011/133`.

[52] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.

[53] B. Wang, X. Wang, R. Xue, and X. Huang. Matrix FHE and its application in optimizing bootstrapping. *The Computer Journal*, page bxy088, 2018.

[54] A. Weber and H. Seidl. On the degree of ambiguity of finite automata. *Theor. Comput. Sci.*, 88(2):325–349, 1991.

[55] S. Yu. Handbook of formal languages, vol. 1. chapter Regular Languages, pages 41–110. Springer-Verlag, Berlin, Heidelberg, 1997.

A Definitions on Regular Expressions and NFA

We recall some standard definitions about regular languages and finite automata [55]. Let Σ be a finite alphabet, and Σ^* the free monoid generated by Σ. A *string* w is an element of Σ^*, which can be written as a finite sequence of symbols $w = w_1 w_2 \cdots w_k$ where $w_1, \ldots, w_k \in \Sigma$, and its *length* is $|w| = k$. The *empty string* is denoted by ϵ, which is the neutral element of Σ^*. The *concatenation* of two strings $u = u_1 \cdots u_m$ and $v = v_1 \cdots v_n$ is a string $uv = u_1 \cdots u_m v_1 \cdots v_n$. A *language* over Σ is a subset of Σ^*. For any languages L and K, we consider the following regular operations: (union) $L \cup K$, (product) $LK = \{uv \mid u \in L, v \in K\}$, and (Kleene star) $L^* = \cup_{i\geq 0} L^i$, where $L^0 = \{\epsilon\}$, and $L^i = LL^{i-1}$ for $i > 0$. *Regular languages* are the smallest class of languages containing the basic languages $\emptyset$, $\{\epsilon\}$, and $\{a_i\}$ for all $a_i \in \Sigma$ that are closed under regular operations.

A *nondeterministic finite automaton (NFA)* over Σ is a quintuple $M = (Q, \Sigma, \delta, Q_I, Q_F)$, where $Q = \{s_1, \ldots, s_n\}$ is a finite set of states, $\delta : Q \times \Sigma \to \wp(Q)$ is a transition function, $Q_I \subseteq Q$ is the set of initial states, and $Q_F \subseteq Q$ is the set of final states. We can extend δ to a function $\delta : Q \times \Sigma^* \to \wp(Q)$ over strings in the natural way. Without loss of generality, we assume that all our NFAs have a single initial state s_1. A string $w \in \Sigma^*$ is *accepted* by an NFA M if $\delta(s_1, w) \cap Q_F \neq \emptyset$. The set of all the strings accepted by an NFA M is called the *language of* M, and it is denoted by $\mathcal{L}(M)$. A *deterministic finite automaton (DFA)* is an NFA such that $\delta(s, a_i)$ is a singleton set for all $s \in Q$ and $a_i \in \Sigma$, and $|Q_I| = 1$.

A *regular expression* over Σ is a formal expression generated by the following grammar rules:

$$\mathsf{RE} \to \epsilon \mid a_i \mid (\mathsf{RE} + \mathsf{RE}) \mid (\mathsf{RE} \cdot \mathsf{RE}) \mid (\mathsf{RE})^*,$$

where a_i ranges over Σ. The operator $*$ takes the highest precedence, followed by $\cdot$, and then by $+$. The parentheses can be omitted when there is no ambiguity. The operator $\cdot$ is usually omitted as well, and concatenations can be written as juxtapositions of regular expressions. For a regular expression e, its *language* $\mathcal{L}(e)$ can be defined inductively as follows:

$$\begin{aligned} &\mathcal{L}(\epsilon) = \{\epsilon\}, && \mathcal{L}(a_i) = \{a_i\}, \\ &\mathcal{L}(e_0 + e_1) = \mathcal{L}(e_0) \cup \mathcal{L}(e_1), && \mathcal{L}(e_0 \cdot e_1) = \{uv \mid u \in \mathcal{L}(e_0), v \in \mathcal{L}(e_1)\}, \\ &\mathcal{L}(e^*) = \cup_{i\geq 0} \mathcal{L}(e)^i, \end{aligned}$$

where a_i ranges over Σ, and e_0, e_1 are regular expressions. For any set R of regular expressions, let $\mathcal{L}(R) = \cup_{e\in R} \mathcal{L}(e)$. It is well known that the languages defined by regular expressions are exactly the regular languages, which are exactly the languages accepted by finite automata.

For any sets R, T of regular expressions, we write RT for the set of regular expressions

$$RT = \{e \cdot f \mid e \in R, f \in T\},$$

and we write $Re = \{f \cdot e \mid f \in R\}$ and $eR = \{e \cdot f \mid f \in R\}$; in particular, $\emptyset T = R\emptyset = \emptyset e = e\emptyset = \emptyset$.

B Proofs

In this section we present proofs that are omitted in the main paper.

Proposition 5.1. *For any $n \geq 1$, if $\mathcal{M}$ is an NFA with $r \leq n$ states, and w a string of length k, the noise vector $\mathbf{e}^{(k)}$ at the end of homomorphic evaluation of encrypted $\mathcal{M}$ on w satisfies the following bounds:*

- *If $\mathcal{M}$ is unambiguous, then $\|\mathbf{e}^{(k)}\|_\infty \leq bnk\chi \log_b q$.*
- *If $\mathcal{M}$ is finitely ambiguous, then $\|\mathbf{e}^{(k)}\|_\infty \leq bnrk\chi \log_b q$.*
- *If $\mathcal{M}$ is infinitely ambiguous, then $\|\mathbf{e}^{(k)}\|_\infty \leq bnk^{\deg(\mathcal{M})+1}\chi \log_b q$.*

Proof. Let $\mathcal{M} = (Q, \Sigma, \delta, \{s_1\}, Q_F)$ be an NFA with r states $s_1, \ldots, s_r$, and for each input symbol $\sigma \in \Sigma$, denote by $\mathbf{M}_\sigma \in \{0,1\}^{n\times n}$ the transition matrix of $\mathcal{M}$ on σ (padded with 0s in the extra columns and rows), where $(\mathbf{M}_\sigma)_{t,s} = 1$ if $t \in \delta(s, \sigma)$, and $(\mathbf{M}_\sigma)_{t,s} = 0$ othewise. For any $t \in Q$ let $\mathcal{M}_t = (Q, \Sigma, \delta, Q, \{t\})$ be the NFA obtained from $\mathcal{M}$ by setting all states to be initial and t the only final state. Notice that $\mathrm{da}(\mathcal{M}_t, l)$ is an upper bound on the total number of paths in $\mathcal{M}$ on a string of length l from any state to t.

Let $w = w_1 \cdots w_k$ be the string to be scanned on $\mathcal{M}$. For all $1 \leq i \leq k$, the encrypted state vector $\mathbf{q}^{(i)}$ after reading w_i is:

$$\mathbf{q}^{(i)} = \sum_{j=0}^{\log_b q} C_{w_i,j}\mathbf{q}_j^{(i-1)} = \beta \mathbf{S}^{-1}\mathbf{M}_{w_i} \cdots \mathbf{M}_{w_1}\mathbf{v} + \mathbf{S}^{-1}(\mathbf{M}_{w_i}\mathbf{e}^{(i-1)} + \sum_{j=0}^{\log_b q} \mathbf{E}_{w_i,j}\mathbf{q}_j^{(i-1)}),$$

where $\mathbf{e}^{(i-1)}$ is the noise term after reading the previous symbol w_{i-1}. As in our assumption, s_1 is always the sole initial state in $\mathcal{M}$, we can set the initial noise $\mathbf{e}^{(0)} = \mathbf{0}$ without leaking any additional information about the NFA $\mathcal{M}$. By expanding all the noise terms, we get

$$\mathbf{e}^{(k)} = \sum_{l=2}^{k} \mathbf{M}_{w_k} \cdots \mathbf{M}_{w_l} \sum_{j=0}^{\log_b q} \mathbf{E}_{w_{l-1},j}\mathbf{q}_j^{(l-2)} + \sum_{j=0}^{\log_b q} \mathbf{E}_{w_k,j}\mathbf{q}_j^{(k-1)}. \tag{7}$$

Notice that, for any symbol $a \in \Sigma$, the (t,s)'th entry of $\mathbf{M}_a$ is 1 if $t \in \delta(s,a)$ and it is 0 otherwise. So the (t,s)'th entry of the product $\mathbf{M}_{w_k} \cdots \mathbf{M}_{w_l}$ counts the number of paths from s to t on the string $w_l \cdots w_k$, where $1 \le l \le k$. Let $\mathbf{1}$ be the vector whose entries are all 1. Then the t'th entry of the vector $\mathbf{M}_{w_k} \cdots \mathbf{M}_{w_l}\mathbf{1}$ counts the total number of paths from an arbitrary state to t on this string, which is at most $\mathrm{da}(\mathcal{M}_t, k-l+1)$. Thus we have

$$\|\mathbf{M}_{w_k} \cdots \mathbf{M}_{w_l} \sum_{j=0}^{\log_b q} \mathbf{E}_{w_{l-1},j}\mathbf{q}_j^{(l-2)}\|_\infty \le bn\chi \log_b q \cdot \max_{t\in Q}\{\mathrm{da}(\mathcal{M}_t, k-l+1)\}.$$

It follows that the final noise vector $\mathbf{e}^{(k)}$ can be bounded by

$$\|\mathbf{e}^{(k)}\|_\infty \le bn\chi \log_b q \cdot \sum_{l=1}^{k-1} \max_{t\in Q}\{\mathrm{da}(\mathcal{M}_t, l)\} + bn\chi \log_b q \tag{8}$$

If $\mathcal{M}$ is unambiguous, then $\mathrm{da}(\mathcal{M}_t, l) \le 1$ for all $t \in Q$ and $l \ge 0$, so

$$\|\mathbf{e}^{(k)}\|_\infty \le bkn\chi \log_b q.$$

If $\mathcal{M}$ is finitely ambiguous, then for all $s, t \in Q$, the number of paths of w from s to t is at most 1 [54]. So $\mathrm{da}(\mathcal{M}_t, l) \le r$ for all $t \in Q$ and $l \ge 0$, and $\mathbf{e}^{(k)}$ can be bounded by

$$\|\mathbf{e}^{(k)}\|_\infty \le bknr\chi \log_b q.$$

For the case where $\mathcal{M}$ is infinitely ambiguous, notice that $\mathrm{da}(\mathcal{M}_t, l) \le l^{\deg(\mathcal{M})}$ for all $l \ge 1$, and we have

$$\begin{aligned}\|\mathbf{e}^{(k)}\|_\infty &\le b\chi \log_b q \sum_{l=1}^{k-1} l^{\deg(\mathcal{M})} + b\chi \log_b q \\ &\le bnk^{\deg(\mathcal{M})+1}\chi \log_b q.\end{aligned}$$

□

C Performance comparisons with HAO15

In this section we present a brief analysis of applying the matrix-FHE scheme of HAO15 [29] to the case of homomorphic evaluation of NFA.

Fix an NFA $\mathcal{M}$ of r states and with an alphabet Σ, and let $\mathbf{M}_\sigma \in \{0,1\}^{r\times r}$ for $\sigma \in \Sigma$ be its transition matrices on symbol σ. Recall the "leveled version" of the HAO15 scheme as described in Section 3.1. To encrypt $\mathcal{M}$ for homomorphic evaluation on any string of length at most k, we sample $k+1$ secret keys sk_i for $i = 0, 1, \ldots, k$, and for each $\sigma \in \Sigma$, we encrypt $\mathbf{M}_\sigma$ with all keys sk_i to get $\mathbf{C}_{\sigma,i} \leftarrow \mathsf{HAO.MatEnc}_{\mathsf{sk}_i}(\mathbf{M}_\sigma)$. We also encrypt the initial state vector $\mathbf{v} = (1, 0, \ldots, 0)^t$ in a ciphertext $\mathbf{c} = \mathsf{HAO.VecEnc}_{\mathsf{sk}_0}(\mathbf{v})$.

To scan $w = w_1 \cdots w_k$ on $\mathcal{M}$, set $\mathbf{c}_0 = \mathbf{c}$ and $\mathbf{c}_i = \mathsf{HAO.Mul}(\mathbf{C}_{w_i,i}, \mathbf{c}_{i-1}) = \mathbf{C}_{w_i,i} \times \mathbf{G}^{-1}(\mathbf{c}_{i-1})$. Then each ciphertext $\mathbf{c}_i$ satisfies $\mathbf{S}_i\mathbf{c}_i = (\prod_{j=i}^{1} \mathbf{M}_{w_j}) \times \mathbf{v} + \mathbf{e}_i$ for some noise vector $\mathbf{e}_i$. By Equation 1, the l_∞ norm of $\mathbf{e}_k$ can be bounded by

$$\|\mathbf{e}_k\|_\infty \le \chi N + \chi N \sum_{l=2}^{k} \mathrm{da}(\mathcal{M}, l) + \chi \mathrm{da}(\mathcal{M}, k),$$

which must be bounded away from $q/4$.

For performance comparison, consider two cases of the ambiguity measures of $\mathcal{M}$:

- $\mathcal{M}$ is finitely ambiguous: We have $\mathrm{da}(\mathcal{M}, l) \le r$ for all $1 \le l \le k$, so w.h.p.

  $$\|\mathbf{e}_k\|_\infty \le \alpha q(n+r)(kr+1)\log q,$$

  where $\alpha = \sqrt{2n}/q$ is the LWE noise parameter. Thus, in the HAO15 scheme we can homomorphically evaluate $\mathcal{M}$ on strings of length $k \le \frac{1}{\alpha(n+r)r\log q}$. For example, for an NFA of up to 1024 states on strings of length up to 137, we need $n = 1024$ and $q = 2^{42}$. On the other hand, using our scheme we can evaluate $\mathcal{M}$ on strings of length $k \le \frac{q}{b^2 n\chi r \log_b q}$. So, using our scheme with the above sets of parameters, we can homomorphically evaluate an NFA of up to 1024 states on strings of length up to 551.

- $\mathcal{M}$ is infinitely ambiguous: We have $\mathrm{da}(\mathcal{M}, l) \le l^{\deg(\mathcal{M})}$, so w.h.p.

  $$\|\mathbf{e}_k\|_\infty \le \alpha q(n+r)\log q \cdot (\sum_{l=1}^{k} l^{\deg(\mathcal{M})} + 1) \le \alpha q(n+r)\log q k^{\deg(\mathcal{M})+1}$$

  Using the same parameters as the above, and assuming that $\deg(\mathcal{M}) = 2$ for the NFA $\mathcal{M}$, we can homomorphically evaluate $\mathcal{M}$ on strings of length up to 65 in the HAO15 scheme, whereas we can homomorphically evaluate $\mathcal{M}$ on strings of length up to 82 in our scheme.

Moreover, the computational complexity of k homomorphic matrix multiplications, assuming naive matrix-vector multiplication of complexity $O(n^2)$, is $O(k(r+n)^2 \log q)$. On the other hand, the complexity of our homomorphic evaluation procedure is $O(kn^2 \log q)$.

D Updated Concrete Security Estimate via Fouque and Kirchner's Analysis

Here we sketch how we estimate the concrete security of MiNTRU using the methods of [34][15]. The attack given in [34] works for any q-ary lattice with a dense sublattice of substantial dimension, not just algebraically structured lattices. A dense sublattice of high dimension allows the BKZ basis reduction algorithm [49] to perform better than when run on a truly random q-ary lattice, under current heuristics.

Recall, the BKZ block reduction algorithm makes oracle calls to an SVP-oracle of dimension b. Therefore, we estimate security as a single call to the SVP oracle. This is known as the *core SVP hardness model* [2]. The best heuristic time complexity for SVP on a rank-n lattice is $2^{c \cdot n(1+o(1))}$, where $c = .292$ for a classical computer and $c = .265$ for a quantum computer, [8] and [35, Section 14.2.10], respectively. BKZ with block size b is expected to return a basis with shortest vector length $\delta_b^r \det(\Lambda)^{1/r}$ where δ_b is the *root Hermite factor* and r is the rank of the input lattice ($b \le r$).

[15] The version of this paper presented at ASIACRYPT 2019 did not take this attack into account. Furthermore, [36] does not attack our original parameter suggestions, $|\chi| = 2\sqrt{n}$ since we used the LWE estimator `https://bitbucket.org/malb/lwe-estimator`. Instead, [36] attacks our toy implementation with binary matrices. Our new estimates show the original entries in Table 2 had roughly 50, and 25 bits of security for $q = 2^{42}$ and $q = 2^{111}$, respectively. The suggested parameters for $q = 2^{883}$ were not secure (less than ten bits of security).

An estimate for the root Hermite factor is, asymptotically, the following function of the block size [16]

$$\delta_b \approx \left(\frac{b}{2\pi e}(\pi b)^{1/b}\right)^{\frac{1}{2(b-1)}}.$$

Given MiNTRU samples $\mathbf{B} = \mathbf{S}^{-1}(\mathbf{G} - \mathbf{E})$, let $\mathbf{D}^t := -\mathbf{S}^{-1}(\mathbf{I}_n - \mathbf{E}_0) \in \mathbb{Z}_q^{n\times n}$ be the negated first square block. We will run BKZ on the lattice $\Lambda_q^\perp([\mathbf{I}|\mathbf{D}]) = \{(\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{Z}^{2n} : \mathbf{x}_1 + \mathbf{D}\mathbf{x}_2 = \mathbf{0} \bmod q.\}$. This is a $2n$-rank lattice with determinant q^n which contains the n short vectors given the columns in $\mathbf{B} := \begin{bmatrix}\mathbf{I}_n - \mathbf{E}_0^t \\ \mathbf{S}^t\end{bmatrix}$, and has determinant roughly L^n where L is the expected length of vectors sampled from χ^{2n}.

D.1 BKZ Predicition

The analysis of [34]'s attack is summarized by Lemma D.1 together with the expected length of the GSO vectors of $\Lambda_q^\perp([\mathbf{I}|\mathbf{D}])$ after running BKZ, [20, Section C]. In short, one makes a prediction of the GSO shape after running BKZ with block-size b and if this contradicts Lemma D.1, then the scheme is broken by BKZ. In more detail, one can expect the volume of the shortest n GSO vectors after running BKZ with block-size b to be, under logarithms, $\frac{\log^2(q)}{16\log(\delta_b)}$ [20, Figure 5]. Therefore, we expect the block-size where we see BKZ perform better than expected to be:

$$\log(\delta_b) = \frac{\log^2(q)}{16n\log(L)}.$$

We estimate the concrete security by finding the smallest block size b which attains the above δ_b. The script "BKZ_security.cpp" at `https://www.dropbox.com/s/10g2nocx3pmyu4t/henfa.zip` performs the above when given n, q, and the expected entry length of χ as input.

Lemma D.1. *([45, Lemma 1]) Let $\Lambda \subset \mathbb{R}^n$ be a full-rank lattice and $r \ge 1$. Then, for any basis $\mathbf{b}_1, \cdots, \mathbf{b}_n$ of Λ with Gram-Schmidt orthogonalization $\widetilde{\mathbf{b}_1}, \cdots, \widetilde{\mathbf{b}_n}$, any r-dimensional sublattice Λ', and any $S \subset \{1, 2, \cdots, n\}$ of size r,*

$$det(\Lambda') \ge \prod_{t_i \in S} \widetilde{\mathbf{b}_{t_i}}.$$

Ключевая структура исследуемого алгоритма

- Регулярные выражения
  - Синтаксический разбор
  - Построение автоматов
  - НКА, с ε-переходами и без.
  - Моделирование переходов через матричную арифметику.
- Криптография на целочисленных решетках
- Гомоморфное шифрование

Регулярные выражения — неформально

- Специальные строки определенного синтаксиса
- Помогают распознать или найти другие строки

Например:

- «`(.*) are (.*) than .*`» шаблон регулярного выражения
- которые соответствует тексту `string`: `”Dogs are smarter than cats”`

Регулярные выражения — формально

Определение

Регулярное выражение *(regular expression) в алфавите* Σ *и задаваемое им множество допустимых слов (язык) в этом же алфавите определяются рекурсивно следующим образом:*

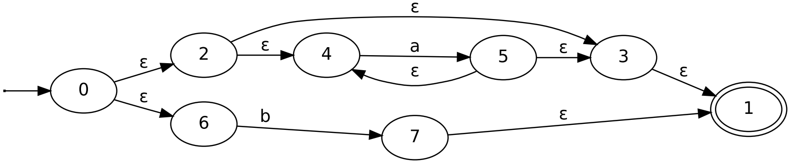
- $\emptyset$ *— регулярное выражение, обозначающее пустое множество слов;*
- ε *— регулярное выражение, задающее пустую строку (пустое слово);*
- *Пусть* $\sigma \in \Sigma$ *— символ из алфавита, тогда* σ *— регулярное выражение, задающее множество, состоящее из этого символа* $\{\sigma\}$*;*
- *Пусть* p, q *— регулярные выражения, задающие языки P и Q соответственно. Тогда*

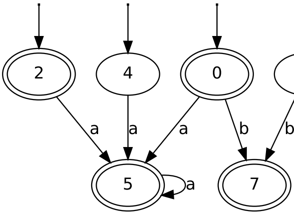
От регулярных выражений к НКА

```
def demo_regex_to_nfa():
    regex = Regex(”(a*|b)”)
    print(regex.get_tree_str())
    enfa = regex.to_epsilon_nfa()
    G = enfa.to_networkx()
    write_dot(G, log.dotprefix + ’demo_regex_to_enfa.dot’)

    nfa = enfa.remove_epsilon_transitions()
    G = nfa.to_networkx()
    write_dot(G, LOG.dotprefix + ’demo_regex_to_nfa_no_eps.dot’)

    mats, state2index, symbol2index = nfa2_zzq_matrices(nfa)


    for sym in symbol2index:
        print(f’Матрица переходов для «{sym}»’)
        print(str(mats[symbol2index[sym]]))

    printverb(’states’,  state2index)
    printverb(’symbols’, symbol2index)
```

```
Operator(Union)
 Operator(Kleene Star)
  Symbol(a)
 Symbol(b)
```

```
def nfa2_zzq_matrices(nfa, N=0, ZZq=ZZ):
    ’’’
    Генерит матрицы переходов
    * для автомата ‘nfa‘
    * в заданном кольце ‘ZZq‘
    * выровненную до квадрата N
    ’’’
    N = max(N, len(nfa.states))
    state2index = {}

    for i, state in enumerate(nfa.states):
        state2index[state] = i

    mats = {}
    symbol2index = {}
    for idx, symbol in enumerate(nfa.symbols):
        symbol2index[symbol] = idx
        mats[idx] = zero_matrix(ZZq, N, N)

    for e_state, symbol, next_state in nfa._transition_function.get_edges():
        i = state2index[e_state]
        j = state2index[next_state]
        mats[symbol2index[symbol]][j, i] = 1

    return mats, state2index, symbol2index
```





Матрица переходов для «b»

```
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[1 0 0 0 0 0 1 0]
```

Индексы для состояний:
`{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7}`

Индексы для символов:
`{b: 0, a: 1}`

Матрица переходов для «a»

```
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[1 0 1 0 1 1 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
```

НКА через матрицу переходов

```python
@dataclass
class RegexByNFA:
    regex_str: str
    ring = ZZ
    N = 0

    def __post_init__(self):
        self.regex = Regex(self.regex_str)
        self.enfa = self.regex.to_epsilon_nfa()
        self.nfa = self.enfa.remove_epsilon_transitions()

        self.N = max(self.N, len(self.nfa.states))

        (self.mats, self.state2index,
            self.symbol2index) = nfa2_zzq_matrices(self.nfa, self.N, self.ring)

    # выставим единицами индексы стартовых состояний
    def set_start_state(self):
        self.state = zero_matrix(self.ring, self.N, 1)
        for state in self.nfa.start_states:
            self.state[self.state2index[state]] = 1

    def eval_symbol_by_nfa(self, idx):
        printv(self.mats[idx])
        self.state = self.mats[idx] * self.state
        pass

    def open_match_text(self, text):
        print(f'Матчинг шаблона «{self.regex_str}» в «{text}»\n\n')
        self.set_start_state()

        for chr_ in text:
            idx = self.symbol2index[chr_]
            self.eval_symbol_by_nfa(idx)

        for state in self.nfa.final_states:
            if self.state[self.state2index[state]] > 0:
                return True
        return False

def demo_nfa_by_matrix():
    r = RegexByNFA('(a*|b)')
    print(r.open_match_text('aab'))
```

Матчинг шаблона «(a*|b)» в «aab»

$$\begin{bmatrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\1&0&1&0&1&1&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{bmatrix} \times \begin{bmatrix}1\\1\\1\\1\\1\\0\\1\\0\end{bmatrix} \Rightarrow \begin{bmatrix}0\\0\\0\\0\\0\\3\\0\\0\end{bmatrix}$$

$$\begin{bmatrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\1&0&1&0&1&1&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{bmatrix} \times \begin{bmatrix}0\\0\\0\\0\\0\\3\\0\\0\end{bmatrix} \Rightarrow \begin{bmatrix}0\\0\\0\\0\\0\\3\\0\\0\end{bmatrix}$$

$$\begin{bmatrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\1&0&0&0&0&0&1&0\end{bmatrix} \times \begin{bmatrix}0\\0\\0\\0\\0\\3\\0\\0\end{bmatrix} \Rightarrow \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

False

GGHLM: Введение

- Работа НКА описывается матрично-векторным произведением
  $\rightarrow$ схема гомоморфна по матрично-векторному произведению;
- Отдельно шифруются матрицы и векторы;
- Дешифрование необходимо только для векторов;
- Описана процедура работы зашифрованного НКА;
- Для обеспечения семантической стойкости схемы при шифровании добавляется шум;

GGHLM: параметры шифрования

- n: максимальное число состояний НКА;
- q, b: параметры перешифрования;
- β: скалирующий коэффициент (отсылка к Регеву), играющий роль верхней границы шума;

GGHLM: генерация ключа

- S — обратимая матрица;
- В качестве собственно ключа выступает пара матриц S, S^{-1};

FHE-Шифрование: Генерация ключа

```python
@dataclass
class GGHLMKeys:
    n: int
    logb: int
    logq: int
    S: object = None
    S_inv: object = None

    def __post_init__(self):
        self.q = 2  ** self.logq
        self.b = 2  ** self.logb
        self.qq = pow(2.0, self.logb * self.logq)


        self.S = zero_matrix(self.ZZq, self.n, self.n)
            fill_matrix_zero_01_random(self.S)
            det = self.S.det()
            if det % 2 == 1:
                break
        self.S_inv = self.S.inverse_of_unit()
```

```
def fill_matrix_zero_01_random(mat):
    for i in range(mat.nrows()):
        for j in range(mat.ncols()):
            mat[i, j] = randrange(2)
GGHLMKeys(
    n=9,
    logb=7,
    logq=6,
    S=[0 0 0 0 1 0 1 0 0]
[1 1 1 0 0 1 1 1 1]
[1 0 0 0 0 1 0 0 0]
[0 0 0 0 1 1 1 1 1]
[0 0 0 0 0 1 0 1 0]
[1 0 1 1 0 0 1 0 0]
[0 1 0 0 1 0 1 1 0]
[0 0 0 1 1 1 1 1 1]
[0 0 1 1 1 0 1 1 1],
    S_inv=[4398046511102 4398046511103
[             1             1             0 43980
[             2             1             0 43980
[             0             0             0 43980
[             1             0             1 43980
[             2             1             0 43980
[             0             0 4398046511103
[4398046511102 4398046511103             0
[4398046511103             0             0
)
```

GGHLM: Шифрование

- M — шифруемая матрица, u — шифруемый вектор;
- E — сгенерированный случайный матричный шум, e — векторный шум;
- Шифр:

$$Enc(M) = \beta S^{-1} M S + S^{-1} E,\ Enc(u) = \beta S^{-1} u + S^{-1} e$$

```
def gghlm_encrypt_nonsquare(key, msg, log_beta):
    beta = 2  ** log_beta
    E = get_random_01_matrix(key.ZZq, msg.nrows(), msg.ncols())
    return beta * key.S_inv * msg + key.S_inv * E

def gghlm_encrypt_square_matrix(key, msg, log_beta):
    beta = 2  ** log_beta
    E = get_random_01_matrix(key.ZZq, msg.nrows(), msg.ncols())
    return  beta * key.S_inv * msg * key.S + key.S_inv * E
```

GGHLM: гомоморфность по произведению

- Зашифрованные матрицы перехода $C_\sigma, \sigma \in \Sigma$, зашифрованный вектор состояний $c = Enc(v)$;
- Работа зашифрованного автомата на входе $w = w_1 ... w_k$:

$$C_{w_k} ... C_{w_1} c$$

- Дешифрование даст

$$Dec(C_{w_k} ... C_{w_1} c) = M_{w_k} ... M_{w_1} v$$

- Зашифрованные матрицы будут умножаться с учётом техники перешифрования (указана ниже), относительно этого произведения схема и гомоморфна;

GGHLM: Дешифрование

Шифротекст:

$$C = S^{-1}(\beta \cdot M + E) = \beta S^{-1} \cdot M + S^{-1}E$$

- Поскольку E — вектор малой нормы, то

$$C \approx \beta S^{-1} \cdot M$$

- «Деления в кольцах — нет» — используем битовый сдвиг:

$$M = S \cdot (C >> \log \beta)$$

```python
def mat_right_shift(mat, shift):
    for i in range(mat.nrows()):
        for j in range(mat.ncols()):
            mat[i, j] >>= shift


def gghlm_decrypt_nonsquare(key, C, log_beta):
    # Раскодируем неквадратные матрицы (вектора)
    res = key.S * C
    mat_right_shift(res, log_beta)
    return res

def gghlm_decrypt_square_matrix(key, C, log_beta):
    # Раскодируем квадратные матрицы
    return gghlm_decrypt_nonsquare(key, C, log_beta) * key.S_inv
```

GGHLM-шифрование + автомат

На входе:

- Матрицы перехода НКА $M_\sigma, \sigma \in \Sigma$;
- Вектором стартовых состояний v;
- На входе строка $w = w_1 ... w_k$

Работа автомата:

$$(M_{w_k} ... M_{w_1})v$$

Соответственно, зашифрованный автомат представляет из себя всё тот же набор, каждый элемент которого зашифрован по процедуре, описанной выше, и происходит перемножение зашифрованных матриц и вектора.

GGHLM-шифрование + автомат

```python
@dataclass
class GGHLMRegexByNFA(RegexByNFA):
    key: GGHLMKeys
    log_beta: int

    def __post_init__(self):
        self.N = max(self.key.n, self.N)
        self.ring = self.key.ZZq

        self.obfs_nfa = [None] * self.sigma * self.key.logq
        for i in range(self.sigma):
            for j in range(self.key.logq):
                entry = i * self.key.logq + j
                self.obfs_nfa[
                    entry
                ] = gghlm_encrypt_square_matrix(
                    self.key,
                    self.mats[i],
                    self.key.logb * j)
```

```
GGHLMRegexByNFA(
    regex_str=’(a*|b)’,
    key=gghlm.GGHLMKeys(
        n=16,
        logb=6,
        logq=7,
        S=[1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0]
[0 0 1 0 0 0 0 1 1 1 1 0 1 1 1 1]
[0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0]
[0 0 0 1 0 1 1 1 0 0 1 1 1 0 1 1]
[1 1 1 0 1 1 0 1 1 0 0 1 1 0 0 0]
[0 1 0 1 1 0 0 1 0 1 1 1 1 1 1 1]
[0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0]
[1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 1]
[1 1 0 0 1 0 0 0 0 1 1 1 1 0 1 1]
[0 1 0 1 0 0 1 1 1 1 0 1 0 0 0 0]
[1 0 1 1 1 0 1 0 1 0 1 1 1 0 0 0]
[1 0 0 0 1 1 0 1 1 0 1 0 0 0 1 1]
[0 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1]
[1 0 0 1 1 1 1 0 0 1 0 1 0 0 1 0]
[1 0 1 1 0 0 0 1 0 1 1 0 1 0 0 0]
[1 1 1 0 1 1 0 1 1 1 1 1 0 0 0 0],
        S_inv=[3871639399736 3464098410290  271693992964  458483613126
[ 696215856970 1471675795221 4369745053504 4075409894459 3515041033971
[2581092933157 3775414443895  181129328643  305655742084  373579240325
[2394303312995 3916921731897  939608392334 1035833348175  288674867524
[2258456316513  554708568968  158488162562 3565983657652 1426393463061
[2020724072670 2733920804200 3922582023417 2496188560357  118866121922
[1664125706904 3803715901495 2971653048043 3090519169965  356598365765
[3464098410290  707536440010 3792395318455 3650888030453 3973524647098
[2513169434917 2094307862431 4188615724861 3769754152375 3141461793646
[2716939929640 1273565592019 2428265062115 2173551943712 3633907155893
[2122609320031 1590541917143 3775414443895 1698087456025 2564112058597
[4143333392700 4324462721343 3299949956208 4194276016381 2682978180519
[ 441502738567 2864107509162 1805632994906 3871639399737 1799972703387
[2411284187555 3237686749487 4335783304383  169808745602  696215856970
[2920710424362 1918838825308 3599945406773  577349735048 4126352518140
[2818825177001  130186704962 2281097482593 1375450839380 1681106581464
    ),
    log_beta=35
)
```

Зашифрованные матрицы перехода

```
{
    0: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0],
    1: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
}
```

```
[
    [2292418065635 2801844302441 1766010954267  288674867525 3667868905013 3175423542766 276788
[1318847924180 2915050132843  549048277449 3543342491572 1816953577946 2326379814754  169808745
[2994294214125 3333911705329 2643356139879  192449911684 2445245936675 3582964532210 331127053
[3713151237175 4375405345024  243392535364 3747112986296 3888620274296  169808745602 315844266
[3169763251245 1267905300498 3962204064056 1267905300497 2139590194591 3684849779574 344711753
[3684849779573  594330609609 1307527341139  594330609609 2377322438436 2139590194592 285278692
[2258456316513 1782991828826 3922582023417 1782991828825 2733920804200 2020724072670 416031426
[3594285115252  809421687371 1194321510737  181129328642 2818825177001  526407111368 363390715
[2722600221159 3979184938618 2303738648673 3350892579889 3769754152375 3141461793646 125658471
[2071666696349  577349735048  390560114885 1205642093777 4194276016382   67923498242 302259567
[2626375265318 2558451767078 3282969081648 3186744125806  390560114886 2801844302441 373579240
[1913178533788 3152782376686  192449911683 3781074735415 2767882553320  543387985927 219053281
[ 299995450563 3135801502125 3673529196533 4392386219583  186789620164 2869767800684 236034156
[3781074735415 2015063781149  328296908165  758479063691 1358469964820 2479207685796  37357924
[2886748675243 3039576546284 1409412588502  526407111369  220751369283 1392431713939  39056011
[1013192182096 2541470892518 2366001855394 3798055609976 2207513692831  730177606090 390560114
    [1579221334155  916967226253 2954672173535 3226366166549  220751369335 3311270539300 310750
[2309398940231 3752773277814 2807504593999 1313187632734  984890724532 1579221334140 123394355
[1052814222791  611311484167  503765945343 2150910777743 1613183083279 2207513692888 353768220
[1613183083252 2071666696348 1788652120374 1262245009036 4245218640089 2105628445500 213392990
[2020724072614 2733920804201  990551015959 4081070185868 4160314267205  832062853397 364522773
[2733920804173  594330609609 1426393463035  950928975323  713196731504 1901857950721 225845631
[3803715901482 1782991828828 4279180389169 2852786926093 2139590194578 1307527341125 237732243
[2801844302433 2903729549803 2026384364182 4352764178927 4364084761976 3888620274289 177733153
[1256584717440 3141461793646 2303738648655 3560323366096 3769754152356 3769754152357 209430786
[ 645273233237 4347103887425 2767882553267  798101104227 1698087455972 3481079284798  56036886
[2428265062104 3396174912050  192449911672 2501848851854 4075409894449 3956543772526  75847906
[ 764139355238 3990505521657 1618843374770 3452777827304  390560114913 1460355212208 301693538
[3073538295341 3345232288370 2960332464939 3299949956081 1375450839316 3039576546220 131884792
[1562240459574  339617491205 3537682200083 2009403489690  407540989478 1715068330616 359428511
[  33961749162 1154699470096 1766010954307  967909850016 4024467270821 3192404417367  49244536
[ 339617491231 2750901678759 1533939001968 2349020980887  662254107874 1137718595564 345843811
    [1511297839152 4330123012864 4313142141592 2258456323091 2801844305731 3362213166219 166412
[2682978182907 3305610247728 1199981804645 2971653052819 2915050135231  566029154397 257543264
[3939562901538 4352764178943 2875428095759 2971653055160 3333911708886  775459941810 110941714
[ 645273235109 1964121157467  622632069027 2496188563995 4375405346842 3198064710665 245656652
[3447117532187 4358424470464  316976321583  950928968289 1267905296956 1228283256315 316976324
[2852786924460  118866121921 3447117534070 1545259581662  594330607948  713196729870 368484977
[4160314266374  356598365767 1545259584098  237732242071 1782991827941 2139590193706 225845631
[1120737720470 1250924425939 1930159407843 2020724071657  809421686866 3316930830262 233770039
[3141461792475 3350892579888 2722600219989 4398046508764 3979184937448 4188615723690 146601550
[1137718592167  492445362247 1715068327217  118866115185  577349731680 4211256887573 332825141
[1850915326355 2354681272355   11320582329 3803715900072 2558451766365 3656548321262 388295998
[ 305655743810 2473547394276 3458438120493  950928978821 3152782378409 4369745055227 316976325
[2988633918521 2847126634601 1726388909546 3922582015257 3135801498046 3469758697730 281316488
[4143333394678 1115077429457 1760350664722 2139590198544 2015063783126 1245264136395 126790530
[3532021911158 3498060159411 2173551946337  237732249094 3039576548909 2767882555946 225845631
```

Зашифрованные переходы

```python
def match_text_encrypted(self, text):

        self.set_start_state()
        self.enc_state = gghlm_encrypt_nonsquare(self.key,
                self.state, self.log_beta)

        print('state: ' + str(self.state.T))
        print('encstate: ' + str(self.enc_state.T))

        # Это может происходить на «вражеской» территории.
        for chr_ in text:
            print(f'chr: {chr_}')
            idx = self.symbol2index[chr_]
            self.eval(idx)
            print('encstate: ' + str(self.enc_state.T))

        self.state  = gghlm_decrypt_nonsquare(self.key,
                      self.enc_state, self.log_beta)
        print('state: ' + str(self.state.T))

        for state in self.nfa.final_states:
            if self.state[self.state2index[state]] > 0:
                return True
        return False
```

```
state: [1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0]
encstate: [3677818636201 4344937057075 951503848730
3357525734013 3671539250297 2316960735896 2381037004746
3027857973997 3393433208345 3752773277816 3916833289842
1732668299529 3305919794922 1690481439294 1099909617023
885879843920]
chr: a
encstate: [1039337337755 147706654157 2740659244127
1756038058514 1630163959396 3905601149556 267920466686
3872987089184 1805093920407 2587292301808 1504289856547
2757101044821 4091582158478 1553615253126 3648731728256
185981005554]
chr: a
encstate: [1966546996231 2378131050888 4382682861015
1844446421787 2682978181648 747158481373 2660337015257
502957332639 2393494700706 3125289533655 3521509939065
289483479964 2248752961639 38004814116 4275945936163
1501594478087]
chr: a
encstate: [ 670879313551 1441218035529 1540946980184
2037974484199 2037704948303 2682978181215 2533923837979
2485137515618 4298317565924 1192165210524 2037435410039
3603449396091 3229600620722 3826896142655 2833110674037
546622437159]
state: [ 0 0 0 0 0 95 0 0 0 0 0 0 0 0 0 0]
True
```

GGHLM: Гомоморфные переходы с перешифрованием

```
def bit_dec(self, state_bd, state):
        for i in range(state.nrows()):
            t = state[i, 0]
            for j in range(self.key.logq):
                state_bd[j, i] = t % self.key.b
```

```
def eval(self, idx):

        work_space = zero_vector(self.ring, self.state.nrows() )
        state_bd = zero_matrix(self.ring, self.key.logq, self.state.nrows() )
        temp_state = zero_vector(self.ring, self.state.nrows() )
        self.bit_dec(state_bd, self.enc_state)
        printverb('enc-state', self.enc_state)
        printverb('state-bd', state_bd)

        for i in range(self.key.logq):
            entry = idx * self.key.logq + i
            work_space = self.obfs_nfa[entry] * state_bd[i]
            temp_state += work_space

        printverb('temp-state', temp_state)

        for i in range(self.enc_state.nrows()):
            self.enc_state[i, 0] = temp_state[i]
```

«enc_state»:

```
[1966546996231]
[2378131050888]
[4382682861015]
[1844446421787]
[2682978181648]
[ 747158481373]
[2660337015257]
[ 502957332639]
[2393494700706]
[3125289533655]
[3521509939065]
[ 289483479964]
[2248752961639]
[  38004814116]
[4275945936163]
[1501594478087]
```

«state_bd»:

```
[ 7  8 23 27 16 29 25 31 34 23 57 28 39 36 35  7]
[ 7  8 23 27 16 29 25 31 34 23 57 28 39 36 35  7]
[ 7  8 23 27 16 29 25 31 34 23 57 28 39 36 35  7]
[ 7  8 23 27 16 29 25 31 34 23 57 28 39 36 35  7]
[ 7  8 23 27 16 29 25 31 34 23 57 28 39 36 35  7]
[ 7  8 23 27 16 29 25 31 34 23 57 28 39 36 35  7]
[ 7  8 23 27 16 29 25 31 34 23 57 28 39 36 35  7]
```

«temp_state»:

```
(670879313551, 1441218035529, 1540946980184, 2037974484199, 2037704948303, 2682978181215, 253392
```

Направления для дальнейшего исследования

- Реализация минимального НКА через производную Брозовского.
- Исследование производительности
  - Теоретической
  - Python-реализации
  - Компиляции через Nuitka
- Альтернативные GSW-схемы шифрования.