

CrypTool Number Field Sieve Extensions

Theodore Winograd, *Member, IEEE*

Abstract—We extend the educational software to support the NFS algorithm. We merge the CrypTool and GGNFS code-base and provide graphical user interfaces for each step of the NFS algorithm. We perform some experiments on the GGNFS implementation as a precursor for future research into the effects of various NFS parameters on performance of the algorithm.

Index Terms—Cryptography, Factoring, Educational Software, CrypTool, GGNFS

I. INTRODUCTION

CrypTool [1] is educational software that aims to assist students in learning about cryptography. CrypTool provides tools for performing encryption, decryption, generating message authentication codes (MACs), digital signatures, and many other cryptographic operations using both modern and historical ciphers. In addition, CrypTool provides some mechanisms for performing cryptanalysis against both classical and modern ciphers, including the RSA algorithm that serves as a bedrock for modern electronic commerce.

CrypTool’s RSA analysis tools include several different factoring algorithms, including trial division, Pollard’s algorithm, the quadratic sieve, and several others[1]. These algorithms are very powerful against small RSA keys (less than about 100 digits), but larger keys require the more efficient Number Field Sieve. While it would be impractical for students to use a single computer to factor large RSA keys, it will be beneficial to provide a user interface for the complex algorithm. This project will extend the CrypTool software by providing support for the Number Field Sieve (NFS).

By extending CrypTool to support NFS, students will be able to see first-hand how the security of RSA depends on the difficulty of factoring large numbers. Similarly, students will also be able to examine how the different NFS parameters can drastically effect the amount of time necessary to factor a large number—anywhere between 15 minutes and two hours for a 50-digit number. Similarly, because students can generate RSA keys of any size in CrypTool, they will be able to

feasibly generate and perform attacks against small RSA keys.

II. OVERVIEW OF NFS

In order to fully understand NFS implementations, it is important to have an understanding of the NFS algorithm. There are five major steps of NFS: [2]

- Polynomial selection
- Sieving
- Mini-factoring
- Linear algebra
- Square root

In polynomial selection, the goal is to generate a polynomial $f(m) \equiv 0 \pmod{n}$ where n is the number to be factored. At first glance, this seems like a simple problem, but the polynomial plays an important role in the algorithm and can have drastic effects on the quality of the results of the sieving step. As such, Murphy [3], Kleinjung [4] and others have described a number of algorithms for selecting valuable polynomials and calculating heuristics indicating their quality. Nevertheless, it is often recommended that users perform some trial sieving to verify the quality of the polynomial.

In the sieving step, the two primary families of algorithms are the line sieve and the lattice sieve as described by Pollard[5]. The experiments performed in this paper and the implementation presented implement the line sieve, but the lattice sieve can be supported with little extra effort. Prior to the sieving step, the following values must be generated:

- *Rational Factor Base*: prime integers up to a certain bound.
- *Algebraic Factor Base*: roots of $f(x)$ modulo p for prime integers up to a certain bound.

Once the factor bases are generated, a sieving interval A is chosen such that $-A < a < A$. Starting with $b = 1$, (a, b) pairs are found such that $a + bm$ is smooth over the rational factor base and such that $a + b\theta$ is smooth over the algebraic factor base. Specifically, this means that all primes occurring in the unique factorization of $a + bm$ must be members of the factor base. Similarly, $a + b\theta$ must have a principal ideal that factors completely into members of the algebraic factor base.

Manuscript received May 8, 2008; revised May 12, 2008

T. Winograd is a graduate student in Computer Engineering at George Mason University, Fairfax, VA 22030 (e-mail: twinogra@gmu.edu)

In the mini-factoring step, a matrix is generated such that each column of the matrix represents an (a, b) pair found in the sieving step. The first entry indicates the sign of $a + bm$ and the following entries represent the factorization of $a + bm$ over the rational factor base and the factorization of the “norm” of $a + b\theta$ over the algebraic factor base. The mini-factoring step can take advantage of many of the factoring algorithms available, including elliptic curve methods (studied by researchers at GMU).

With the matrix generated, we perform the matrix step, in which we “find a dependency among the binary vectors corresponding to the (a, b) pairs” produced in the sieving step. These dependencies represent $a + bm$ pairs that result in a square in \mathbb{Z} . There are several algorithms available for performing the matrix step, primarily those described by Wiedemann and Lanczos.

In the final step, the square root step, we process the (a, b) pairs to generate $\sqrt{\prod_{(a,b)}(a + bm)}$ and $\sqrt{\prod_{(a,b)}(a + b\theta)}$. The resulting values can be used in the congruent squares factoring algorithm. If $x^2 \equiv y^2 \pmod n$ then we can find the factors of n (p and q) by computing $\gcd(n, x \pm y)$.

III. PREVIOUS WORK

Since the advent of the Number Field Sieve, many organizations and researchers have been working on the algorithm, improving its efficiency [3][6][7][8][4][9] or developing implementations of the algorithm [10][11][12][13][14][15][16]. Several previous publications are of particular importance for this project.

In 2006, a team from GMU presented an implementation of the Elliptic Curve method of factoring at the CHES conference [17]. One of the primary uses of this implementation would be to improve the mini-factoring step of the NFS algorithm. In 2007, Chandana Anand (GMU) developed C and MAGMA implementations of the Wiedemann and Block Wiedemann algorithms for the matrix step of the number field sieve [18].

In Fall 2007, as part of my thesis research, I evaluated a number of different readily available implementations of NFS:

- Chris Monico: GNU General Number Field Sieve (GGNFS) [11]
- Per Leslie Jensen: Pleslie’s General Number Field Sieve (pGNFS) [10]
- Chris Card: factor-by-gnfs [12]
- Jason Papadopoulos: msieve [13]

From the results of my investigations, I found msieve to be the most efficient and freely licensed implementa-

tion of NFS. In addition, msieve is actively developed, with the most recent release on January 13, 2008. Msieve has been used by the NFSNet project and other researchers as one of the primary tools for factoring numbers as large as 518 bits.

IV. OVERVIEW OF CRYPTTOOL

CrypTool is a C++ application written using Microsoft Visual C++ .NET 2003 and the Perl scripting language for the Windows Platform. There is a port of CrypTool for Linux [19] in progress, but it is not advanced enough for the purposes of this project. As mentioned previously, CrypTool provides tools for performing cryptographic operations using classical and modern ciphers as well as tools for performing cryptanalysis. All of these tools are available from the CrypTool menu bar. When a particular operation is desired, CrypTool will open a dialog box for the user to provide input about the key or the type of operation to be performed, allowing for a graphical user interface to what are otherwise “black box” algorithms.

The CrypTool source code relies on the Microsoft Foundation Classes (MFC) GUI library for interacting with the user, preventing it from easily being ported to other platforms. Nevertheless, the MFC library is a very straightforward interface for developing GUI applications. By relying on a single main menu to launch dialog boxes for each application, the CrypTool developers have ensured that it is relatively painless to introduce new functionality into CrypTool: new functionality need only be added as a menu item and the appropriate dialog box can be instantiated.

To aid in distribution, all of the libraries that CrypTool requires are statically linked to the CrypTool application, precluding users from downloading additional software—or even running a software installer. As such, any extensions to CrypTool should aim to keep with this tradition.

V. GGNFS AND MSIEVE

Both msieve and GGNFS rely on Murphy’s α approximation to generate polynomials. Both implementations provide line sieves, but msieve’s line sieve uses the bucket sort algorithm outlined by Aoki and Ueda.[6] For the linear algebra step, both applications provide an implementation of the Lanczos algorithm.

In testing performed to compare the GGNFS and msieve implementations, GGNFS was found to factor a 00-digit number in 10 hours (via its lattice sieve implementation) while msieve factored the same number in a little over four days via its line sieve implementation. When compared to a commercial NFS implementation provided by the MAGMA computational algebra system,

which took over six days to complete, GGNFS proved the most efficient candidate. Nevertheless, the GGNFS source code is very convoluted with little documentation and (at the time) had not been updated in several years.

Msieve, on the other hand, had a number of benefits over GGNFS:

- msieve relies on its own multi-precision implementation
- msieve does not perform lattice-based sieving as suggested by portions of the readme
- msieve implements a bucket-sort version of the line sieve, as outlined by Kazumaro Aoki and Hiroki Ueda. [6]
- like GGNFS, msieve relies on no additional libraries beyond its own multi-precision library
- msieve is actively developed (the latest release is 1.35 on 4/14/2008)
- msieve is written to be a library that can be included by other applications (the msieve application is a demonstration of how these library functions may be used)
- the msieve code is well-commented

The primary drawback of the msieve implementation is that it is hard-coded to factor only numbers larger than 97 digits via NFS. Nevertheless, the code changes necessary to disable this check and provide parameters for smaller numbers were negligible. As a testament to msieve's performance, the msieve Web site indicates that it is used by the NFSNet project to perform the final stages of NFS.

Partway through this project, I discovered that Jason Papadopoulos has been granted commit access to the GGNFS subversion repository with the intent of merging the GGNFS and msieve code-bases. Once complete, the more efficient GGNFS sieving code will work seamlessly with the more efficient msieve matrix processing code.

VI. IMPLEMENTATION RESULTS

The original goal of the implementation phase of this project was to integrate both the msieve and GGNFS code-base into CrypTool, allowing users to choose which algorithms to use with different steps of the NFS. In particular, the NFS extension would allow users to perform the polynomial selection step separate from the rest of the algorithm. Because polynomial selection is a non-deterministic function, this will allow users to record and re-use the results of this step before running the rest of the algorithm. Similarly, users would be able to see first hand how polynomial selection can affect the rest of the algorithm. Users would also have the ability to input polynomials generated using other tools to test

their effectiveness. For the rest of the algorithm, users will be able to supply the NFS parameters described in Table I to monitor how the choice of parameters affects the amount of time spent running the NFS algorithm.

TABLE I
NFS IMPLEMENTATION PARAMETERS

General Parameters	
n	The number to factor
$f(x)$	The polynomial NFS will use
m	A value of x at which $f(x) = 0$
P_{max}	The largest prime to be used in each factor base
RFB	The size of the rational factor base
RFB_{max}	The largest value of the rational factor base
AFB	The size of the algebraic factor base
AFB_{max}	The largest value of the algebraic factor base
QCB	The size of the quadratic character base (optional)
Polynomial Selection Parameters	
α	The maximum allowed value of Murphy's α approximation in selecting the polynomial
d	The degree of the polynomial (usually 3 or 5)
Line Sieving Parameters	
A	For line sieving, the sieving interval such that $-A < a < A$

Ultimately, users would be allowed to specify the algorithms used for each step of the NFS algorithm in addition to specifying NFS parameters. Users would be able to store the results of each intermediate step and perform analyses of the performance of each option.

Because msieve has no external library dependencies and provides a Win32 implementation from their Web site, it was chosen as the first NFS implementation to integrate with CrypTool. Nevertheless, several road-blocks were encountered while working on this phase of the project. In particular, the Win32 implementation of msieve requires functions specific to Microsoft Visual Studio 2005, a compiler that CrypTool does not support and cannot compile in. At the beginning of this project, it seemed relatively simple to port the msieve code to Visual Studio .NET 2003, but the resulting binary consistently failed to factor different 40-digit numbers (see Figure 1).

After testing in Linux it became apparent that something was wrong with the msieve implementation. Either the test vectors used earlier to verify msieve was capable of factoring smaller numbers were a special case, or a change had been introduced into the msieve code base that introduced these errors. When the latest version of msieve was tested, it continued to fail. As such, the decision was made to go with GGNFS, which would introduce its own problems.

After some research on the GGNFS Web site[20], I was able to find Visual Studio project files for the GGNFS suite that support Visual C++ 6.0, Visual C++

Fig. 1. Msieve Error

```

C:\WINDOWS\system32\cmd.exe
sieving in progress (press Ctrl-C to pause)
h = 1240, 190616 complete / 280870 batched relations (need 190000)
sieving in progress (press Ctrl-C to pause)
h = 17854, 2097543 complete / 22 batched relations (need 1960815)815)
sieving in progress (press Ctrl-C to pause)
h = 41216, 3083322 complete / 4 batched relations (need 2856355)6355)
error: too many filtering passes
C:\msieve-1.33\build_vc8\win32\debug>gnfs -n -r 500000
factor base: found 20001 rational and 20062 algebraic entries
sieving in progress (press Ctrl-C to pause)
h = 738, 560811 complete / 27 batched relations (need 500000)000)
sieving in progress (press Ctrl-C to pause)
h = 3317, 2155242 complete / 94 batched relations (need 2127878)878)
sieving in progress (press Ctrl-C to pause)
h = 5156, 3184580 complete / 198 batched relations (need 2918352)52)
error: too many filtering passes
C:\msieve-1.33\build_vc8\win32\debug>

```

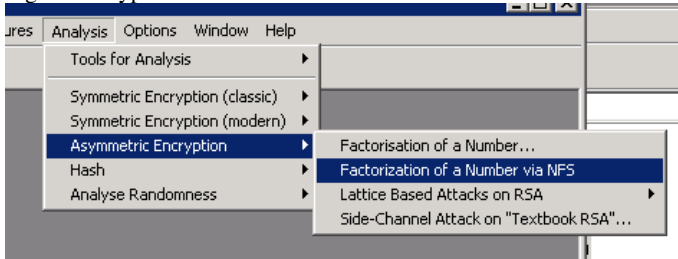
.NET 2003, Visual C++ 2005 and Visual C++ 2008. While work on GGNFS has begun anew under Jason Papadopoulos, there were still a number of issues to deal with:

- GGNFS relies on the GMP 4.1.4 library which is newer than the version used by CrypTool
- The Windows patches for GMP 4.1.4 are not easily available [21]
- The Visual Studio project files did not function out-of-the-box

After a good deal of work, it was possible to compile and run GGNFS and GMP 4.1.4 in Windows. Several rounds of testing resulted in the same output and findings as testing in Linux. As such, it became possible to work on the CrypTool merge.

A CrypTool menu item was created (see Figure 2) and a Dialog Box was developed for NFS (see Figure 3), polynomial selection (see Figure 4), factor base generation (see Figure 5) and sieving (see Figure 6).

Fig. 2. CrypTool NFS Menu



This led to the final road blocks in the phase of the project: updating CrypTool's GMP library and integrating the polynomial selection code. Updating the GMP library proved straightforward once all references to the older library file were removed from the CrypTool project and replaced with the newer 4.1.4 version. CrypTool continued to function without any issues, indicating the two GMP versions are binary compatible (except for the functions required by GGNFS).

The final hurdle in this phase was converting the

Fig. 3. NFS Dialog

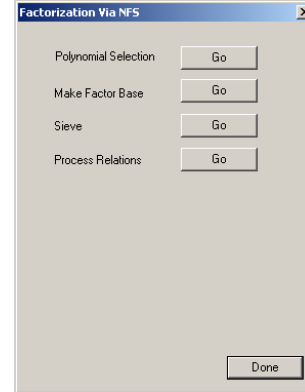
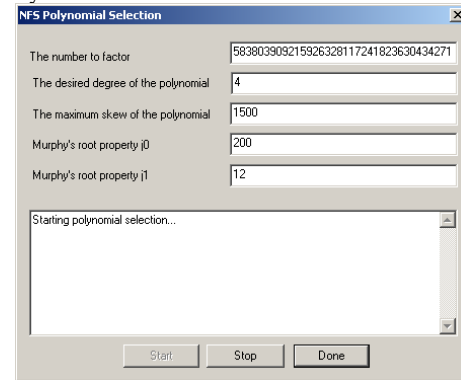


Fig. 4. Polynomial Selection



polynomial selection code to C++ in such a way that it would compile, run and output could be displayed to the dialog. As such, the current version of the CrypTool extension includes support for every NFS step up to matrix generation. The biggest obstacle in compiling the polynomial selection code was getting the LN2 identifier and other mathematical constants to compile. In the GGNFS project, the compiler does not complain about these identifiers—nor does the GCC compiler in Linux. After much searching, I found there is a define that must be set to include math.h constants. Even though it is not explicitly set in the GCC or main GGNFS Visual C++ project, somehow it is enabled when those projects are compiled. Nevertheless, once the appropriate define was enabled, CrypTool began to correctly compile the polynomial selection code.

Finally, to allow users to cancel the polynomial selection process, the polynomial selection code was modified to be launched in a separate thread, allowing the CrypTool user interface to remain responsive while searching for appropriate polynomials. Once polynomial selection was complete, merging future steps with CrypTool had fewer issues.

Fig. 5. Factor Base

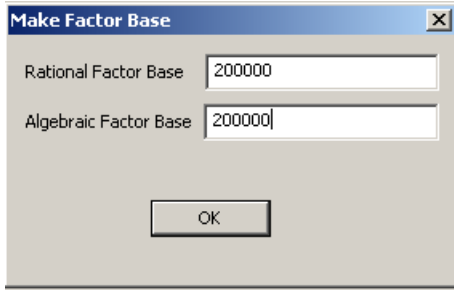
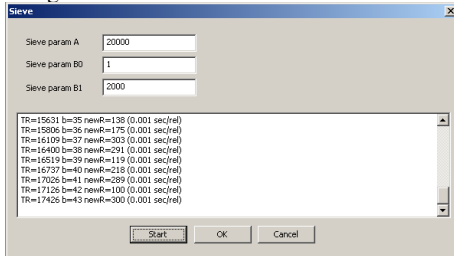


Fig. 6. Sieving



VII. EXPERIMENT RESULTS

Due to mishaps in the implementation phase of the project, experiments were performed using the GGNFS suite of programs on Enigma—without the CrypTool interface as originally planned. The Enigma machine has the following properties:

- Owned and operated by GMU
- Quad Xeon 2.8 GHz processor
- 4 GB RAM
- Several hundred GB of hard disk space

As such, Enigma provides a platform on which these experiments can easily be repeated.

The GGNFS suite of programs divides the NFS process into a number of steps. Each step is defined as its own application with its own set of inputs and parameters. This allows users of the GGNFS suite to tune a large number of NFS parameters to the specific number and polynomial that will be used. Table II outlines the parameters supported by the basic GGNFS polynomial selection function. GGNFS also provides an implementation of Franke’s polynomial selection algorithm, but it is not suitable for numbers smaller than 98 digits. In addition, there is no documentation describing the parameters that the improved polynomial selection code accepts.

The second phase of the GGNFS suite generates the factor base. The parameters supported for factor base generated are outlined in Table III.

The third phase of the GGNFS suite performs sieving. As GGNFS provides two sieving implementations (line and lattice), there are slightly different parameters

TABLE II
POLYNOMIAL SELECTION PARAMETERS

n	The number to factor
d	The desired degree of the polynomial
j_0 and j_1	Murphy’s sieving-for-root properties
$maxs1$	Threshold for the initial polynomial test
$maxskew$	The maximum acceptable skew for the polynomial
$lc1$	The maximum relative size of the leading coefficient
lcp	The number of primes to choose from for leading coefficient divisors
$leave$	The number of bits of the leading coefficient that will be random

TABLE III
FACTOR BASE PARAMETERS

n	The number to factor
f	The polynomial chosen for NFS
m	The value at which $f(m) = 0 \pmod n$
rl	The maximum size of the rational factor base
al	The maximum size of the algebraic factor base
mpr	The max large rational prime for large-prime variation
mpa	The max large algebraic prime for large-prime variation
p	The number of large rational and algebraic primes

available for each. Table IV shows the parameters for each.

TABLE IV
SIEVING PARAMETERS

n	The number to factor
f	The polynomial chosen for NFS
m	The value at which $f(m) = 0 \pmod n$
$(r a)\lambda$	How far from perfect sieve values to look for good relations.
q_0	The initial value of q for the lattice sieve
$qintsize$	The q range size for the lattice sieve
a_0	The initial value of a for the line sieve
a_1	The final value of a for the line sieve
b_0	The initial value of b for the line sieve
b_1	The final value of b for the line sieve

Once sieving is complete, the GGNFS suite provides an application that will process the relations from one or more sieve runs into a format suitable for generating the matrix. The matrix step consists of three applications: one for building, pruning, and solving the matrix. The parameters available during the matrix step are given in Table V.

The final step of the GGNFS suite, as with the final step of the NFS algorithm, is the square root step. As with the other steps, GGNFS provides several parameters that may be of use, identified in Table VI.

As there is a relationship between skew and the results of the sieving step, an initial line of experiments would

TABLE V
MATRIX PARAMETERS

<i>minff</i>	The minimum number of FFs
<i>maxrelsinff</i>	The maximum relation-set weight
<i>wt</i>	The weight factor determining the sparseness of the matrix.

TABLE VI
SQUARE ROOT PARAMETERS

<i>depnum</i>	The specific dependency to try from the matrix solution
<i>knowndiv</i>	The product of known small divisors of n

be to perform some tests based on the skew results of the polynomial. In particular, Monico states in the documentation that most skews are between 1500 or 2000. Similarly, j_0 and j_1 are used to find "nearby" polynomials that may have more small roots, providing smoother ideals. Nevertheless, there will be some trade-off between the amount of time spent in polynomial selection and the amount of time spent performing the rest of the algorithm.

In addition, the effects of sieving parameters (e.g., a and b) were also evaluated, aside from the obvious values affecting the performance of the sieving step. It is, nevertheless, expected that lattice sieving will perform far better than line sieving.

According to the GGNFS documentation, the two most important parameters are *maxrelssinff* and *wt*. *maxrelsinff* defines the maximum number of relations per relation-set when preparing the matrix. It is expected that this value and the sparseness of the matrix will have a direct impact on the performance of the matrix solution step.

The test procedures will be as follows:

- 1) Generate multiple n 's of different sizes (40-digit, 80-digit, more if time allows)
- 2) Generate multiple polynomials for each value of n with different parameters
- 3) Generate multiple sieving runs for each polynomial with different parameters
- 4) Generate multiple matrices for each sieving run with different parameters
- 5) Tabulate the time spent in each step
- 6) Analyze the results

Tests were developed for a 40-digit number, but they were not effectual because with the appropriate sieving parameters, NFS can factor a 40-digit number in less than 30 seconds. As such, more tests were performed on the 80-digit number.

Table VII shows the results of different sieving parameters on a polynomial generated to have a skew of 1500 for an 80-digit number. The results column indicates the result of the matrix generation step, showing the number of columns generated against the minimum required by the algorithm. Table VIII shows the results for a polynomial with a skew of 1625. Higher skews are not provided because higher skew polynomials were not found even when specified.

TABLE VII
RESULTS OF A 1500 SKEW POLYNOMIAL

A_0	A_1	B_0	B_1	Time	Results
$-2 * 10^6$	$2 * 10^6$	1	2000	4m51s	2233/71537
$-2 * 10^6$	$2 * 10^6$	1	20000	33m43s	3949/71537
$-2 * 10^6$	$2 * 10^6$	1	100000	182m15	5748/71537

TABLE VIII
RESULTS OF A 1625 SKEW POLYNOMIAL

A_0	A_1	B_0	B_1	Time	Relts
$-2 * 10^6$	$2 * 10^6$	1	2000	5m47s	2935/71297
$-2 * 10^6$	$2 * 10^6$	1	20000	35m0s	5805/71297
$-2 * 10^6$	$2 * 10^6$	1	100000	185m56s	8829/71297

As expected, the polynomial with a higher skew produces better results. Nevertheless, the effect of the higher skew is almost negligible. More interesting results are shown in Table IX, which illustrates the effects of altering the sieving parameter a vs. the sieving parameter b .

TABLE IX
RESULTS OF ALTERING THE VALUE OF a VS b

A_0	A_1	B_0	B_1	Time	Relts
$-4 * 10^6$	$4 * 10^6$	1	2000	18m0s	5062/71297
$-6 * 10^6$	$6 * 10^6$	1	2000	19m30s	6862/71297
$-8 * 10^6$	$8 * 10^6$	1	2000	26m14s	8529/71297
$-10 * 10^6$	$10 * 10^6$	1	2000	43m21s	10103/71297
$-2 * 10^7$	$2 * 10^7$	1	2000	41m17s	17017/71297
$-4 * 10^7$	$4 * 10^7$	1	4000	138m54s	57503/71297
$-4 * 10^7$	$4 * 10^7$	1	8000	253m97s	71297/71297
$-6 * 10^7$	$6 * 10^7$	1	2000	103m53s	41891/71297
$-8 * 10^7$	$8 * 10^7$	1	2000	130m2s	52969/71297

Because sieving must find pairs $a+bm$ that are smooth over the factor base, it is interesting to notice that the size of a has little effect on the number of relations produced per second vs. the size of b . When running the sieving step on Enigma, low values of b result in 0.002 relations per second. With the default value of a at $2 * 10^6$, the value of b increases much faster, resulting in as much as 0.014 relations per second for higher values of b . In contrast, a can be as high as $8 * 10^7$ without effecting

the amount of time per relation. At $8 * 10^7$, the amount of time necessary for each relation begins to increase slightly, but not at the rate that higher values of b provide. This shows that it is possible to perform line sieving with larger values of a in a significantly reduced amount of time. In fact, when compared to the 60m required for lattice sieving to process this 80-digit number, it seems it may be possible to achieve similar performance (perhaps only twice as slow) with line sieving.

It was also interesting to note that the parameters for matrix generation and processing had no effect on the algorithm. In fact, if an invalid parameter was supplied (either too small or too large) GGNFS would ignore it and calculate a better, more optimal parameter.

VIII. FUTURE WORK

The mini-factoring step can take advantage of many of the factoring algorithms available, including elliptic curve methods (studied by researchers at GMU) and the quadratic sieve. Finally, alternate algorithms are available for the linear algebra step as well, including the Wiedemann and Lanczos algorithms. Once the first implementation of the CrypTool extension is complete, it is hoped that multiple algorithms will be implemented for each step of NFS, allowing users of CrypTool to gain a full understanding of the benefits and drawbacks of the choices available for each step of NFS.

It will also be beneficial to perform some additional experiments, testing the effects of the rational and algebraic factor bases on sieving as well as comparing the effects found in this project to larger numbers—perhaps as high as 250 digits.

Another interesting avenue of research would be to test the effects of variable values of a and b on sieving. In a paper by Scott Contini in 2007, he suggested performing line sieving over more precise regions instead of a large rectangle based solely on the values of a and b . Combined with the findings in this paper, it may be possible to configure a line sieve implementation that can perform on par with a lattice sieve implementation.

Follow-on work for the implementation portion of this project will be to fully implement support for different algorithms as well as a number of improvements to the interface:

- Provide users with a range of numbers to choose from
- Provide users with an estimate of the number of relations necessary
- Provide users with default values appropriate for the size of the number chosen
- Integrate NFS with the other factoring algorithms

Additional future work would include modifying the GGNFS code to integrate better with external libraries. In its current incarnation, the polynomial selection code has 50 functions that are specific to the polyselect project (and thus needed to be copied to the CrypTool implementation and modified to run within the CrypTool environment). Many of these functions have some form of output, resulting in only necessary output being sent to the CrypTool user. All of the GGNFS tools have similar numbers of functions (procrels.c is 1300 lines long), making the merge process very time consuming for each tool. Re-factoring the code should also aid in the efforts to merge the GGNFS and msieve code-base.

IX. CONCLUSION

The CrypTool educational software was expanded to provide support for the Number Field Sieve factoring algorithm. Users may now compare the effects of NFS parameters on factoring and gain a better understanding of the algorithm as a whole by seeing it operate first hand, adding to the educational experience. In this project, I provided some insight into how these parameters affect the performance of the algorithm to serve as a starting point for future endeavors. Nevertheless, the project was not entirely successful because roadblocks in the implementation phase caused a refocusing of efforts away from what seemed—at the outset—to be the best NFS implementation available. However, the new path of merging solely with GGNFS follows the stated plan of the GGNFS and msieve projects, preventing future efforts on this project from replacing msieve support with GGNFS in the future.

REFERENCES

- [1] CrypTool. [Online]. Available: <http://www.cryptool.com/>
- [2] M. E. Briggs, “An introduction to the general number field sieve,” Masters Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, April 1998.
- [3] B. Murphy, “Polynomial selection for the number field sieve integer factorisation algorithm,” Ph.D. dissertation, Australian National University, Jul 1999.
- [4] T. Kleinjung, “On polynomial selection for the general number field sieve,” *Mathematics of Computation*, vol. 75, no. 256, pp. 2037–2047, Jun 2006.
- [5] J. M. Pollard, “The lattice sieve,” in *The Development of the Number Field Sieve*, ser. Lecture Notes in Mathematics, vol. 1554. Springer Berlin / Heidelberg, 1993, pp. 43–49.
- [6] K. Aoki and H. Ueda, “Sieving using bucket sort,” in *Advances in Cryptology - ASIACRYPT 2004*, ser. Lecture Notes in Computer Science, vol. 3329. Springer Berlin, Nov 2004, pp. 92–102.
- [7] L. T. Yang, L. Xu, and M. Lin, “Integer factorization by a parallel GNFS algorithm for public key cryptosystems,” in *Embedded Software and Systems*, ser. Lecture Notes in Computer Science, vol. 3820. Springer Berlin, Nov 2005, pp. 683–695.

- [8] N. Guo, L. T. Yang, M. Lin, and J. P. Quinn, "A parallel GNFS integrated with the block wiedeemanns algorithm for integer factorization," in *2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*. IEEE Computer Society, Sep 2006, pp. 45–50.
- [9] J. Franke and T. Kleinjung, "Continued fractions and lattice sieving," in *Special-Purpose Hardware for Attacking Cryptographic Systems SHARCS*, 2005.
- [10] P. L. Jensen, "Integer factorization," Masters Thesis, University of Copenhagen, Copenhagen, Denmark, Dec 2005.
- [11] C. Monico. GNU general number field sieve. [Online]. Available: <http://www.math.ttu.edu/~cmonico/software/ggnfs>
- [12] C. Card. factor-by-gnfs. [Online]. Available: <https://sourceforge.net/projects/factor-by-gnfs/>
- [13] J. Papadopoulos. Integer factorization source code. [Online]. Available: <http://www.boo.net/~jasonp/qs.html>
- [14] T. Izu, J. Kogure, and T. Shimoyama. (2007, Sep) CAIRN 3: An FPGA implementation of the sieving step with the lattice sieving. [Online]. Available: http://www.hyperelliptic.org/tanja/SHARCS/talks07/sharcs2007_cairn3_3.pdf
- [15] K. Aoki, J. Franke, T. Kleinjung, A. K. Lenstra, and D. A. Osvik, "A kilobit special number field sieve factorization," in *Advances in Cryptology ASIACRYPT 2007*, ser. Lecture Notes in Computer Science, vol. 4833. Springer Berlin, Nov 2007, pp. 1–12.
- [16] S. Cavallar, B. Dodson, A. Lenstra, P. Leyland, W. Lioen, P. L. Montgomery, and B. Murphy, "Factorization of RSA-140 using the number field sieve," in *Advances in Cryptology - ASIACRYPT99*, ser. Lecture Notes in Computer Science, vol. 1716. Springer Berlin, 1999, pp. 195–207.
- [17] K. Gaj, S. Kwon, P. Baier, P. Kohlbrenner, H. L. Le, M. Khaleeluddin, and R. Bachimanchi, "Implementing the elliptic curve method of factoring in reconfigurable hardware," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, ser. Lecture Notes in Computer Science, vol. 4249. Springer Berlin / Heidelberg, Oct 2006, pp. 119–133.
- [18] C. Anand, "Factoring of large numbers using number field sieve – the matrix step," May 2007.
- [19] CrypTooLinux. [Online]. Available: <http://www.cryptoolinux.net>
- [20] C. Monico and J. Papadopoulos. GNU general number field sieve. [Online]. Available: <http://sourceforge.net/projects/ggnfs>
- [21] Wingmp. [Online]. Available: <http://na-inet.jp/na/bnc/wingmp.html>