A Tutorial on High Performance Computing applied to Cryptanalysis

(invited talk abstract)

Antoine Joux

DGA and
Université de Versailles Saint-Quentin-en-Yvelines, Laboratoire PRISM,
45 avenue des États-Unis, F-78035 Versailles Cedex, France
antoine.joux@m4x.org

Abstract. Cryptology and computers have a long common history; in fact, some of the early computers were created as cryptanalytic tools. The development of faster and widely deployed computers also had a great impact on cryptology, allowing modern cryptography to become a practical tool. Today, both computers and cryptology are not only practical, but they have became ubiquitous tools. In truth, computing devices incorporating cryptography features range from very small low-end devices to supercomputer, going through all possible intermediate sizes; these devices include both general purpose computing devices and specific, often embedded, processors which enable computing and security features in hundreds of technological objects.
In this invited talk, we mostly consider the cryptanalytic side of things, where it is fair to use very large amounts of computing power to break cryptographic primitives or protocols. As a consequence, demonstrating the feasibility of new cryptanalytic methods often requires large scale computations. Most articles describing such cryptanalyses usually focus on the mathematical or algorithmic advances and gloss over the implementation details, giving only sufficient data to show that the computations are feasible. The goal of the present abstract is to give an idea of the difficulty facing implementers of large scale cryptanalytic attacks.

Computers and cryptanalysis have a long common history. This is well-emphasized by the location of this Eurocrypt conference located near Bletchley Park, the home of the UK code-breaking during World War II. In particular, the park features a working replica of the first digital computer, the Colossus and of the Turing-Welchman Bombe, which was initially developed for cryptanalytic purposes. The organization of the park itself reflects the duality of computers and cryptanalysis. Indeed, the park hosts two museums, the "National Codes and Ciphers Centre" and the "National Museum of Computing".

Even if computers and other computing devices have become general purpose tools in the present days, they still have a lot in common with cryptography. Today, almost all computing devices, from credit cards to high-end computers implement some cryptographic functionality and cryptography is an essential tool for securing the digital world. Moreover, most of the recent cryptographic advances rely on the enhanced performances of modern computing devices.

On the cryptanalytic side, we encounter a similar situation. Having faster and bigger computers allows cryptanalysts to run huge computations which would not have been possible in their wildest dreams a few decades ago.

This article discuss this cryptanalytic application of supercomputers. In Section 1, we classify the typical cryptanalytic applications. In Section 2, we describe the hardware context of the last decade and discuss some possible evolutions. Section 3 explains the practical issues that can be encountered while managing the necessary computations to set new cryptanalysis records. Finally, Section 4 describes some algorithm challenges that need to be solved to efficiently use the potential power of forthcoming computers.

1 Typical cryptanalytic applications

The applications of high-performance computing to cryptanalysis are numerous and varied. They range from attacks which are "embarrassingly-parallel" and can trivially use a large distributed computing power to algorithms which are essentially sequential by nature and are very difficult to adapt to take advantage of the power of supercomputers.

The easiest case of embarrassingly parallel computations contains brute-force attacks and their variants. In this case, each task can run completely independently of the others, it only needs to receive a small amount of input data (such as a plaintext/ciphertext pair) and a description of the part of the key space it should work on. Note that this description is not enough necessary and, especially when the control loop is loose, simply letting each task try a random subset of the key might even be preferable. Some other attacks, such as differential collision searches on hash functions are also of an embarrassingly parallel nature [4].

A slightly harder class of computations which can be parallelized in a reasonable straightforward way, but require communications to send back some partial results in a centralized place. This centralized place then redistributes the values in order to conclude the computation. This is typically the case of parallelized collision-finding algorithms [14, 21].

The next important class of problem contains the sieving-based index calculus algorithm for factoring [1, 5, 15] and discrete logarithms [11–13]. In this class, the largest phase of the computation (the sieving phase) is embarrassingly parallel, however, it produces a large amount of data which needs to be collected in a centralized place. Note that this amount of data is small compared to the magnitude of the computation but it is still a difficult task to centralize this data without introducing errors. The next phase consists in transforming this

data into a linear system of equations and then in solving this system. This offers much more difficulty than the initial computation. Currently, this task is achieve by first reducing the size of the system using ad'hoc heuristics called, structured Gaussian Elimination. This is usually done on a small number of processor, but the computational cost required here is low enough and this is not a problem. The reduced system is then solved using an iterative linear solver such as the Lanczos or Wiedemann algorithms. The main problem is that these algorithms can be distributed but require a large amount of communications between the individual tasks. As a consequence, even when using the block Wiedemann variant [6, 20] which lowers the amount of communications, this is usually the computational bottleneck.

Finally, some cryptanalyses rely on algorithmic tasks for which no satisfactory parallel descriptions are known. This is the case for many advanced algorithms used in cryptanalysis (see Section 4). Note that even in the best cases, writing record-breaking codes is a very specific programming activity, which rarely follows the tenants of modern software engineering. The reason for this discrepancy is that the use of modern programming features has a cost in terms of performance, which is rarely acceptable in this specific context.

2 Hardware context

During the last decades of the twentiest century, the speed of processor increased at the steady rhythm. More precisely, clock rates were at the MHz level in the 80s and raised to the GHz level in the 2000s. This increased the performance of individual processors and permitted to do bigger computations while using at most a small amount of parallelism. However, the clock rates of processors are no longer increasing and the additional computing power of recent processors come from their ability to perform more computations in parallel. This capability is obtained either by allowing the machines to work on larger data types, by giving CPUs the ability to parallelize micro-instructions or by building multi-core processors. As a consequence, despite the stopped growth of clock rate, the raw computing power of processors is still increasing steadily. However, taking advantage of this power for cryptanalytic tasks requires much more effort on the programmer's part.

At the same time, the amount of memory available in modern machines has increased considerably. In the 80s, 64 Kbytes of memory for a personal computer was above standard, in the present days, the equivalent would be around 8 Gbytes. However, on modern processors, accessing memory is proportionally more expensive. To palliate this problem, designers have added several levels of memory-cache that greatly increase the memory accesses as long as they remain reasonably localized. This is also an important constraint since in this model some algorithmic techniques such as sieving are considerably slowed down.

Where personal computers are concerned, the main processor(s) is no longer the only available computing ressource. Indeed, with the development of 3D-games, graphics cards have progressively been transformed into massively paral-

lel computing ressources, capable of performing quite general computations. As a consequence, it has become worthwhile to consider their potential as computing devices in massive computations.

Above the personal computer scale, the development of cloud computing is offering a new opportunity to run medium-scale computation at a moderate cost. At the present time, these infrastructures seem are more suited to embarrassingly parallel tasks than to communication bounded computations. One advantage of using cloud-computing, emphasized in [16], it that it gives a simple metric to compare computations: their monetary cost.

Finally, turning to supercomputers, it is interesting to see that, even at this large scale, many computers among the most powerful are built by assembling many high-end "personal computers" tied together by a high-performance network. As a consequence, running embarrassingly-parallel task on such computers does not require much programming beyond the initial work of writing the program for a general purpose computer. It also means that the previous considerations about parallelism and memory accesses remain true. Of course, thanks to the high-performance network, it is possible to perform tasks that require a fairly high amount of communications. However, despite this improve performance, communications often remain the bottleneck point for algorithms which are not straightforward to parallelize.

Another possibility to perform very large computations is to consider the use of specific hardware. However, the cost of building such hardware is high. As a consequence, many papers [10, 18, 19] dealing with specific hardware remain theoretical and aim at finding the limit of feasible computations. A notable exception is the development of the DES-Cracker [9].

3 Running record computations

Once a new cryptanalytic algorithm has been discovered or improved and implemented, running the algorithm to set some record computation is a nice way to demonstrate the potential of the algorithm. This being said, one could easily imagine that this final step of performing the computation is just a routine task. Unfortunately, this is not the case and running record computations is a difficult and tedious task.

The first step is to obtain the necessary computing resources. This can be easy if the computation only requires dozens of desktop computers for a few weeks or become a real nightmare for people trying to run a large scale computation by recruiting tens of thousands of computers on the Internet. The easiest approach for computations that requires significantly more power than a dozen of desktop computers is to apply for computing time one or several supercomputers. Throughout the world, there exists several supercomputing organizations that let researchers apply for computing power.

The next step, once computing power has been granted, is to port the computing code to the computers that have been made available. Even when great care has been taken to write portable code in the first place, there are always

specific "features" that call for modifications. This is especially true for complex computations that have successive computing phases. Indeed, in that case, one often discovers that one of the "negligible" phases of the computation does not scale well and needs a complete rewriting to run correctly for the record being considered.

Once all this preparation has been settled and, contrary to what might be expected, the really hard part of the computation starts. Indeed, while large computations may become routine when series of similar computations are performed[1], record computations never are. A first problem is that by going to larger sizes, one often triggers unexpected bugs, with rare probability of occurrences. This may lead the program to make several passes over the same search space, which not only wastes computing power but may trigger other bugs when unexpected data collisions are encountered later on. Another possible consequence is that some intermediate computational data may contain corrupted information. While benign in some application such as brute force, incorrect data may cause major failures in other cases. For example, if a single incorrect equation is added to a large linear system, then any hope of recovering the solution is lost. Note that corrupted data is not always a consequence of coding bugs, supercomputers are often experimental machines which may suffer from occasional hardware problem and the sheer scale of the computation also makes physical corruption of data, whether in memory or on disk, possible.

As a consequence, when programming with record computations in mind, it is essential to add extra robustness in the processing. A good practice is to check intermediate computational results whenever this can be done at a reasonable cost. Such checking should use independently written code and should do the verification at the highest achievable level of mathematical abstraction. For example, before performing the iterative linear algebra step of a sieving algorithm it is very good practice to pull back the equation to the mathematical group being considered and check them on this group.

Also note that closely monitoring the computation is a must: processes may get stuck, they may fail to restart after maintenance. To make it short, when running record computations, one should always expect the unexpected.

4 Algorithmic challenges

As computations grow bigger, the relevant metric to measure the computation is shifting. We can no longer focus on running time and ignore other parameters. Of course, running time has never been a perfect metric but it still gave a good approximation of the efficiency of algorithms. With modern supercomputers, the pictures is much more complicated. First, the gap between the cost of time and memory is growing bigger. Second, another, very important parameter should be taken into account: the cost of communication between the parts of the supercomputer.

[1] A typical example is the weather forecast computations which despite their large scale become routine once the production code becomes stable enough.

Combining all the relevant parameters is not easy because there the parameters are not independent. Indeed, programs which require a lot of memory cannot store their data locally in a single node and are going to use larger amount of communications.

As a consequence, in order to use supercomputers to their full power, new algorithms are becoming necessary. These new algorithms should be designed with new metrics in mind. Basically, processes should be as independent of each others as possible and memory use should be limited to fit within local memory (or even better within cache memory).

Of course, embarrassingly parallel tasks are not going to be a problem. However, there are many more algorithms which need to be adapted or improved to become more efficient on supercomputer. To give some example, let us mention iterative linear algebra [17, 20], structured Gaussian elimination, computation of Gröbner bases [8], SAT solvers [7], collision-search techniques [14, 21], large-scale lattice reduction [2], generalized birthday algorithms [3], ...

5 Conclusion

Performing cryptanalytic records computation is a very efficient tool to understand the concrete security level of cryptographic primitive and this should remain true in the future. In particular, such computations can be used to benchmark lower security levels. Indeed, on the one hand, many low-end cryptographic devices rely on a 80-bit security level. On the other hand, the current fastest computer can perform more than 2^{73} floating-point instructions per year. As a consequence, since the figures are getting close, studying the evolution of record computations is essential in order to decide when to phase out such low-end systems before they become insufficiently secure.

References

1. Kazumaro Aoki, Jens Franke, Thorsten Kleinjung, Arjen K. Lenstra, and Dag Arne Osvik. A kilobit special number field sieve factorization. In Kaoru Kurosawa, editor, *ASIACRYPT'2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2007.
2. Werner Backes and Susanne Wetzel. Parallel lattice basis reduction - the road to many-core. In Parimala Thulasiraman, Laurence Tianruo Yang, Qiwen Pan, Xingang Liu, Yaw-Chung Chen, Yo-Ping Huang, Lin huang Chang, Che-Lun Hung, Che-Rung Lee, Justin Y. Shi, and Ying Zhang, editors, *13th IEEE International Conference on High Performance Computing & Communication*, pages 417–424. IEEE, 2011.
3. Daniel J. Bernstein. Better price-performance ratios for generalized birthday attacks. Available from `http://cr.yp.to/rumba20/genbday-20070904.pdf`, 2007.
4. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of sha-0 and reduced sha-1. In Ronald Cramer, editor, *EUROCRYPT'2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005.

5. Richard P. Brent. Recent progress and prospects for integer factorisation algorithms. In Ding-Zhu Du, Peter Eades, Vladimir Estivill-Castro, Xuemin Lin, and Arun Sharma, editors, *Computing and Combinatorics, 6th Annual International Conference*, volume 1858 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.
6. Don Coppersmith. Solving linear equations over GF(2) via block Wiedemann algorithm. *Mathematics of Computation*, 62:333–350, 1994.
7. Youssef Hamadi (editor). Special issue on parallel SAT solving. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:203–262, 2009.
8. Jean-Charles Faugère and Sylvain Lachartre. Parallel Gaussian elimination for Gröbner bases computations in finite fields. In Marc Moreno Maza and Jean-Louis Roch, editors, *Proceedings of the 4th International Workshop on Parallel Symbolic Computation*, pages 89–97. ACM, 2010.
9. Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. O'Reilly & Associates, Inc., 1998.
10. Jens Franke, Thorsten Kleinjung, Christof Paar, Jan Pelzl, Christine Priplata, and Colin Stahlke. Shark: A realizable special hardware sieving device for factoring 1024-bit integers. In Josyula R. Rao and Berk Sunar, editors, *CHES'2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 2005.
11. Takuya Hayashi, Naoyuki Shinohara, Lihua Wang, Shin'ichiro Matsuo, Masaaki Shirase, and Tsuyoshi Takagi. Solving a 676-bit discrete logarithm problem in GF(3^{6n}). In Phong Q. Nguyen and David Pointcheval, editors, *PKC'2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 351–367. Springer, 2010.
12. Antoine Joux and Reynald Lercier. The function field sieve in the medium prime case. In Serge Vaudenay, editor, *EUROCRYPT'2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 254–270. Springer, 2006.
13. Antoine Joux, Reynald Lercier, Nigel P. Smart, and Frederik Vercauteren. The number field sieve in the medium prime case. In Cynthia Dwork, editor, *CRYPTO'2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 326–344. Springer, 2006.
14. Antoine Joux and Stefan Lucks. Improved generic algorithms for 3-collisions. In Mitsuru Matsui, editor, *ASIACRYPT'2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 347–363. Springer, 2009.
15. Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit rsa modulus. In Tal Rabin, editor, *CRYPTO'2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350. Springer, 2010.
16. Thorsten Kleinjung, Arjen K. Lenstra, Dan Page, and Nigel P. Smart. Using the cloud to determine key strengths. *IACR Cryptology ePrint Archive*, page 254, 2011.
17. Thorsten Kleinjung, Lucas Nussbaum, and Emmanuel Thomé. Using a grid platform for solving large sparse linear systems over gf(2). In *Proceedings of the 2010 11th IEEE/ACM International Conference on Grid Computing*, pages 161–168. IEEE, 2010.
18. Arjen K. Lenstra and Adi Shamir. Analysis and optimization of the twinkle factoring device. In Bart Preneel, editor, *EUROCRYPT'2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 35–52. Springer, 2000.
19. Adi Shamir and Eran Tromer. Factoring large number with the twirl device. In Dan Boneh, editor, *CRYPTO'2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2003.

20. Emmanuel Thomé. Subquadratic computation of vector generating polynomials and improvement of the block wiedemann algorithm. *J. Symb. Comput.*, 33(5):757–775, 2002.
21. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.