On the cost of cryptanalytic attacks

Jean-Philippe Aumasson

FHNW, 5210 Windisch, Switzerland

Abstract. This note discusses the complexity evaluation of cryptanalytic attacks, with the example of exhaustive key search, illustrated with several ciphers from the eSTREAM project. A measure is proposed to evaluate the effective computational cost of cryptanalytic algorithms, based on the observation that the standard one is not precise enough.

1 Introduction

The notion of *time complexity* arose in complexity theory in order to evaluate the hardness of computational problems; informally, the time complexity of a problem (as a mathematical object) is defined as the minimum (over all Turing machines) of the maximum (over all problem instances) number of transitions required by a Turing machine to solve it, as a function of the length (in bits) of the machine's input. Time complexity is generally expressed using asymptotic notations; for example, stating that an algorithm has time complexity $\mathcal{O}(n^2)$ means that there exists $P \in \mathbb{Z}[X]/(X^3)$ and a Turing machine solving any instance of the problem in less than $f(n)$ transitions, with f the polynomial function induced by P, for all input of length $n \in \mathbb{N}$.

Extending trivially this notion to algorithms has no sense, and the simplified notion of *algorithmic complexity* is used instead; atomic operations are arbitrary defined (memory access, comparison of values, fixed-length arithmetical operations, *etc.*), and are counted in the algorithm, without regard to a particular calculus model, in terms of the algorithm's arguments. Generally, instead of their length, their value is used (leading to an exponential gap, which brings confusion), and the operations running in constant time are considered as atomic, since captured by the asymptotic notations. It is essentially easy to evaluate asymptotic algorithmic complexities, but their knowledge is not sufficient to compare the effective efficiency of algorithms, since huge multiplicative or additive constants may be "hidden". For example, integer multiplication of n-bit numbers runs in $\mathcal{O}(n^2)$ with the naive algorithm, and $\mathcal{O}(n \log n \log\log n)$ with a FFT-based algorithm; however the naive method is faster for $n < 100$.

Cryptographers are generally far more interested in effective computation times than in theoretical asymptotic algorithmic complexities. In particular, using asymptotics has no sense for fixed-length-parameters ciphers – indeed $\mathcal{O}(2^k) = \mathcal{O}(1)$ for a k-bit key cipher. Clearly, the model of the Turing machine is not good to evaluation the hardness of cryptanalytic attacks: the atomic operation is a machine transition, which not trivially translates on modern computers, and evaluating rigorously the cost of an attack in this model is complicated. Using the algorithmic complexity to evaluate an attack cost is not good either, for two main reasons: first, the operations arbitrary chosen as elementary may not be so in an implementation of the algorithm; secondly the implemented algorithm may be much different from the pseudo-code version, by using for example precomputed tables, factorised instructions, loops simplifications, *etc.* Hence the evaluation of the computation of an attack must not only rely on the pseudo-code algorithm. Instead, a lower bound is generally given, in terms of non-atomic

operations, like "run key and IV setup", "compute 500 Kb of keystream", or "compute the corresponding value", but a precise statement of the real cost lacks, and so this bound is often confused with the approximate cost of the attack.

So as to give a more precise evaluation of the cost of a cryptanalytic algorithm, we choose to consider as atomic time unit a CPU cycle, and suggest to estimate a lower bound on number of cycles required to run the algorithm considered (on a serial machine with "reasonnable" memory and CPU), and define this measure as the *effective cost* of the algorithm. One may object that the number of cycles required to run some optimised algorithm changes from one architecture to another; we do not target the minimum number of cycles of an implementation; but the minimum *necessary* in any implementation on a "reasonnable" machine.

We prefer to use the term "cost" instead of "complexity", since the latter refers to an intrinsic quality of some abstract object, like a decision problem, whereas "cost" translates the properties of its relation with some concrete object (here computers), and thus fits better in our situation. The computational cost must not be confused with the effective execution time, neither with the notion of security (*cf.* [4, 7]).

2 Exhaustive key search

It is obvious that for a cipher with k-bit secret key and full key domain, an algorithm performing the EKS requires more than 2^k operations, which translates, on a computer, to a number of clock cycles above 2^k, assuming that a key trial takes at least one cycle. The value 2^k is indeed the complexity of the *generic* algorithm, where the operation "try a key" is assumed atomic, and we abusively talk about "an attack running in time 2^k", whereas 2^k is only a *lower bound* on the effective cost. However, most of the stream ciphers require a non-trivial key setup stage, whose cost should not be neglected. One may also consider the cost of computing the few keystream bits, but for half of the keys less only one bit of keystream are necessary, thus EKS cost should only consider one bit of output to estimate a lower bound on the complexity (except when the ciphers output blocks of bits).

Table 1 shows, for several ciphers [18, 6, 5, 8, 3, 11, 15, 10, 9, 14, 12, 19, 8] of the eSTREAM project [1],

- the key length (in bits),
- the IV length (in bits),
- the effective cost evaluated of key setup and IV setup, along with the induced least number of cycles of EKS, in binary logarithmic units (between [...]).
- the same information for the latest experimental results [2] over a 2 137 MHz Intel Core 2 Duo with most recent implementation (reference 20061108).

We also present several ciphers for which no effective cost is given, sorted by decreasing key length. Note that the results of software implementations of hardware ciphers (with superscript †) may not be relevant, since software efficiency is not the main target. Note that, although exhaustive search is done for fixed IV's, the IV setup process must be performed for each key, since always comes after the key setup, and depends on it.

We justify our evaluations of the effective cost:

- HC-256: key setup first computes 2 560 32-bit values iteratively, each requiring 20 operations (arithmetic or logic), then updates the internal state 4 096 times, each update requiring 27 operations. Counting at least one cycle per 32-bit value computed, and also at least one per state update, key setup requires at least 6 656 cycles.

Table 1. Cost of EKS for eSTREAM ciphers.

Cipher	*Key*	*IV*	*Effective cost*	*Measures*
Dragon	256	128	–	1 365 [266.4]
HC-256	256	128	6 656 [268.7]	83 617 [272.3]
Phelix	256	128	–	1 320 [266.3]
Py	256	128	323 [264.3]	6 039 [268.5]
Salsa20	256	64	1 [256.0]	40 [261.5]
ABCv3	128	128	–	204 084 [145.6]
Grain-128†	128	128	256 [136.0]	463 [136.8]
Lex-v1	128	128	300 [136.2]	473 [136.8]
Mickey-128†	128	128	416 [136.7]	31 880 [142.9]
Rabbit	128	64	12 [131.6]	754 [137.5]
Sosemanuk	128	64	800 [137.6]	1 615 [138.6]
Hermes8-80†	80	64	–	2 584 [91.3]
Trivium†	80	80	288 [88.1]	616 [89.2]

- Py: key setup consists in 323 loops of a block of instruction including a S-box look-up and several bitwise operations. We assume that each iteration takes more than one cycle.
- Salsa20: key setup is straightforward and only consist in a few simple operations, so we make no specific assumption on the minimal number of cycles requires.
- Grain-128: the feedback registers are clocked 256 times before generating keystream. We count at least 256 cycles for this operation.
- Lex: Rijndael's key schedule is run during key setup, which requires about 300 with the best implementations.
- Mickey-128: key setup performs 416 updates of the internal states, each requiring 1 764 simple bitwise operations. We count at least one cycle per update.
- Rabbit: 8 variables of the inner state are computed, then the cipher called 4 times, so at least 12 cycles are necessary.
- Sosemanuk: key setup performs 48 calls to the SERPENT block cipher; regarding to the best implementations of SERPENT, at least 800 cycles are necessary.
- Trivium: each bit of the internal state requires four iterations of a loop of about 20 operations. We assume that each of the 288 bits computed requires more than on cycle.

Table 1 shows that the real cost of exhaustive search is almost always much greater than 2^k (the smallest multiplicative gap is for Salsa20, only $2^{5.3}$, the highest for ABCv3, $2^{17.6}$). Following the experimental results, about 2 000 Salsa20 keys are tried while only one of HC-256 is set up. HC-256 indeed suffers of an extremely long key setup, but provides an EKS cost increased to $2^{268.7}$, although 2^{256} would clearly be high enough. But the gain is not overkill for ciphers with 80-bit and 128-bit keys. The average gap (logqrithmically) induced by our estimations of the effective cost is 2^9. As a consequence, an attack against a k-bit-key cipher with effective cost less than 2^{k+a}, for some $a > 0$, would "break" the cipher if exhaustive search runs in more than 2^{k+a} cycles.

3 Other attacks

Not only exhaustive search can be subject to the effective cost evaluation, we give a few examples hereafter.

- The key recovery attack against LEX in [20] requires 20 000 keystream bytes for each of 2^{61} random IV's: according to our previous results, computing all those keystreams requires

more than $2^{77.8}$ cycles (with AES implementation running at 15 cycles per byte, and key schedule in 300 cycles), whereas the complexity given by the authors is 2^{61}. If we follow the benchmarks' results, the attack would take more than $2^{78.2}$.
- The distinguishing attack on Py in [17] runs in $t \cdot 2^{84.7}$, with t the cost of key and IV setup. Our evaluation gives a global cost of $2^{92.7}$; benchmarks give $2^{97.2}$.
- The chosen-ciphertext attack against Mosquito in [13] of "complexity" 2^{70} requires 64 clockings of the cipher for 2^{70} keys (no particular key setup is specified for this cipher), giving an effective cost greater than 2^{76}.

Concretely, a 4 GHz processor performs 2^{32} cycles per second, thus would take 2^{44} seconds to perform 2^{76} cycles, that is, more than 550 000 years. But recall that brute force attacks can be parallelised, and use non trivial techniques to run faster (see for example [16]).

4 Conclusion

The new measure presented seems far more realistic and precise than the standard one, and shows that several ciphers with the same key length can provide much different EKS computation times. Like almost all security arguments, our hardness statements rely on assumptions; no strict rule defines the evaluation of the effective cost, since the appreciation of what costs at least one cycle is quite arbitrary. However our estimations are sound with the experimental results, and the improvement margin remains quite large for most of the ciphers. Evaluating the effective cost of other cryptanalytic algorithms may lead to surprising results.

References

1. eSTREAM, the ECRYPT Stream Cipher Project. Available at `http://www.ecrypt.eu.org/stream/`.
2. Daniel J. Bernstein. Notes on the ECRYPT stream cipher project (eSTREAM). Timings available at `http://cr.yp.to/streamciphers/#timings`.
3. Daniel J. Bernstein. Salsa20. eSTREAM [1], Report 2005/025, 2005.
4. Daniel J. Bernstein. Understanding brute force, 2005. Document ID: 73e92f5b71793b498288efe81fe55dee. `http://cr.yp.to/papers.html#bruteforce`.
5. Eli Biham and Jennifer Seberry. Py (Roo): A fast and secure stream cipher using rolling arrays, 2005.
6. Alex Biryukov. A new 128 bit key stream cipher : LEX. eSTREAM [1], Report 2005/013, 2005.
7. Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. *Lecture Notes in Computer Science*, 1976:1–??, 2000.
8. Martin Boesgaard, Mette Vesterager, Thomas Christensen, and Erik Zenner. The stream cipher Rabbit. eSTREAM [1], Report 2005/024, 2005.
9. Christophe De Cannière and Bart Preneel. Trivium - a stream cipher construction inspired by block cipher design principles. eSTREAM [1], Report 2005/021, 2005.
10. Stephan Lucks Doug Whiting, Bruce Schneier and Frédéric Muller. Phelix - fast encryption and authentication in a single cryptographic primitive. eSTREAM [1], Report 2005/020, 2005.
11. Come Berbain *et al.* Sosemanuk, a fast software-oriented stream cipher. eSTREAM [1], Report 2005/027, 2005.
12. Martin Hell, Thomas Johansson, and Willi Meier. Grain - a stream cipher for constrained environments. eSTREAM [1], Report 2005/010, 2005.
13. Antoine Joux and Frédéric Muller. Chosen-ciphertext attacks against MOSQUITO. In *FSE'06*, 2006.
14. Ulrich Kaiser. Hermes8. eSTREAM [1], Report 2005/012, 2005.
15. William Millan Joanne Fuller Leonie Simpson1 Ed Dawson Hoonjae Lee Kevin Chen, Matt Henricksen and Sangjae Moon. Dragon: A fast word based stream cipher. eSTREAM [1], Report 2005/006, 2005.
16. Philippe Oeschlin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO'03*, 2003.
17. Souradyuti Paul, Bart Preneel, and Gautham Sekar. Distinguishing attacks on the stream cipher Py. eSTREAM [1], Report 2005/081, 2005.

18. Ilya Kizhvatov Vladimir Anashin, Andrey Bogdanov and Sandeep Kumar. ABC - a new fast flexible stream cipher specification, version 3. eSTREAM [1], Report 2005/001, 2005.
19. Hongjun Wu. A new stream cipher HC-256. eSTREAM [1], Report 2005/011, 2005.
20. Hongjun Wu and Bart Preneel. Attacking the iv setup of stream cipher LEX. eSTREAM [1], Report 2006/050, 2006.