

# Password Hashing Competition second round candidates – tests report (version 0)

MILAN BROZ

gmazyland@gmail.com

March 24, 2015

## Abstract

The second round of Password Hashing Competition presents some tweaked and some new versions of submitted algorithms. This report tries to run several specific test cases to verify usability of the provided candidates. Tests include portability (use on different platform), illustration of dependence between run time and memory use and input parameters. The last test covers specific key-derivation application.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Algorithms . . . . .	2
1.2	Environment . . . . .	2
1.3	Code repository and compilation fixes . . . . .	2
1.4	PHS function prototype . . . . .	2
1.5	Parameters . . . . .	2
<b>2</b>	<b>TEST1: Test vectors and endianness test</b>	<b>3</b>
<b>3</b>	<b>TEST2: Dieharder test</b>	<b>3</b>
<b>4</b>	<b>Parameters tests</b>	<b>4</b>
4.1	TEST3: Variable memory cost . . . . .	4
4.2	TEST4: Variable time cost . . . . .	4
<b>5</b>	<b>Password input and hash output length test</b>	<b>9</b>
5.1	TEST5: Impact of input length to run time (with fixed costs) . . . . .	9
5.2	TEST6: Impact of output lengths to run time (with fixed costs) . . . . .	9
<b>6</b>	<b>TEST7: Real use case test for KDF mode</b>	<b>14</b>
<b>7</b>	<b>Conclusion</b>	<b>14</b>

## 1 Introduction

This report extends benchmarks of the Password Hashing Competition(PHC) [1] candidates for the first round [2] and covers tweaked versions of the submitted candidates.

This report is not meant as a performance test comparison (submitted candidates are often reference implementations written for better readability and not performance). The main intention of tests is to compare limits and generic behavior of algorithms and also uncover technical problems for function deployment.

### 1.1 Algorithms

For algorithms are selected all submitted candidates for the second PHC round: Argon [3] (with new submission Argon2i and Argon2d), battcrypt [4], Catena framework [5] (in two instances Catena-Dragonfly and Catena-Butterfly, Lyra2 [6], MAKWA [7], Parallel [8], POMELO [9], Pufferfish [10] and yescrypt [11].

Candidates MAKWA and Parallel do not have memory cost attribute, so tests are limited.

In addition, I selected subset of candidates (Argon, Battcrypt, Catena-Butterfly, Lyra2 and yescrypt) and run specific key-derivation use case test (intended use is to replace key derivation function in disk encryption application). (The selection criteria was is mainly usability as a generic KDF, portability and stability and reusing existing proven cryptographic primitives. The detailed selection criteria is out of scope of this report.)

### 1.2 Environment

Tests were run on 64bit Linux operating system (both kernel and userspace is fully 64bit), on two platforms

- X86-64: Lenovo X230 notebook with i5 CPU and 16GB RAM with AES-NI and SSE instructions available (typical user notebook configuration with additional memory)
- PPC64: PowerPC7 server in big-endian mode, 128GB RAM (for endianness and portability tests)

Tests intentionally do not use parallel attribute.

### 1.3 Code repository and compilation fixes

The repository includes copy of submitted source code of PHC algorithms and compiles static library for each algorithm.

Variable cost tests use special utility that measures difference in used memory using *getrusage()* system call. So tests measure real additional memory that running process requests from operating system.

Run time measurement is using *clock\_gettime(CLOCK\_MONOTONIC)* on the Linux platform.

The test run as a special forked process started for each test separately, tests are repeated for 5 times and arithmetic mean (for the time) or maximum (for the memory) of measurements is used.

Algorithms that provides optimized variant are tested separately (only AES-NI and SSE variants). Because AES-NI and SSE instructions are not usable on PowerPC system, only reference implementations run (and compile) there.

The repository with source code and logs is available on GitHub <https://github.com/mbroz/PHCTest>.

All changes to submitted code are tracked by quilt, see *patches* sub-directory in git (separate for every algorithm). There are many tweaks to code to allow compilation on different platform (PPC64) but also other problems (many of them are caused by MSVC vs gcc compiler).

Important fixes:

- Argon: unsupported includes and pre-processor macros for gcc, unconditional use of non-standard 128bit integer types.
- Argon: tweaked PHS() function (added m.thread option) is not C-only compatible and cause crashes.
- Argon: several mixed unsigned int / uint32\_t / size\_t parameters in attributes are incompatible and cause crashes.
- Argon2i: not compilable in submitted version, even with version from GitHub it often crashes. (Removed from KDF test for this reason.)
- Argon, battcrypt, Parallel: missing extern C specifier for PHS()
- Makefiles: removed -march=native for reference versions (not portable to PPC64)
- Makefiles: removed SSE, AVX, etc options for generic reference versions (not portable to PPC64)

### 1.4 PHS function prototype

The PHC competition [1] required all candidates to provide this C function interface:

```
int PHS(void *out, size_t outlen,          /* output */
        const void *in, size_t inlen,      /* input */
        const void *salt, size_t saltlen,  /* salt */
        unsigned int t_cost,               /* time cost */
        unsigned int m_cost);              /* memory cost */
```

Following tests are using only this provided function as the interface to algorithms.

### 1.5 Parameters

The candidate functions do not implement the API parameters consistently. Some functions implement only one of the cost parameters (memory or time) and the scale differs significantly.

The following tests take the password hash function as a black-box, supposing that invalid or unsupported parameters combination is detected and function returns an error.

## 2 TEST1: Test vectors and endianness test

Some algorithms do not provide their own test vectors (specifically not for PHS() interface), so a different strategy was used: all variable input tests produced logs of input parameters and output hash in the first round and these logs are used to determine that algorithms still provide the same output (including optimized variant).

Vectors are stored in *hash\_vectors* sub-directory.

For x86-64 platform:

- all algorithms pass tests in repeated run.
- optimized variants match the reference output (Argon-AESNI, Lyra2-SSE, POMELO-SSE and yescrypt-SSE). Note that this applies *only with applied fixes* for PHC() function mentioned in section 1.3.

For PPC64 (big-endian) platform:

- Optimized variants (Argon-AESNI, Lyra2-SSE, POMELO-SSE and yescrypt-SSE) cannot be tested.
- Vectors passed: battcrypt, MAKWA, Parallel, yescrypt.
- Vectors *fail*: Argon, Argon2i, Argon2d, Catena-Butterfly, Catena-Dragonfly, Lyra2 (producing zeroed output), POMELO, Pufferfish.

This test apparently shows that most of the implementations is not portable yet. One clear issue is that initial hash includes often algorithm parameters in initial round in platform native mode instead of using conversion macros before hashing.

## 3 TEST2: Dieharder test

The output of hashing functions should pass basic randomness tests. Fail in these tests usually indicates some serious internal problem. (While passing tests indicate in fact nothing:-)

The test generates 32-bytes hashes of consecutive little-endian integer (4 bytes) with fixed 16 byte salt. Output is written into file (for time reasons the file size is limited to 400MB). All algorithms use minimal values of memory cost and time cost.

The dieharder testsuite is run with file input generator (dieharder -a -f file -g 201).

No problems were detected in this test (test was run only on x86-64 platform).

## 4 Parameters tests

The following tests use one variable attribute (memory cost, time cost, input or output length) while all other parameters are fixed.

Note the attributes limits are not yet aligned to suggested attributes in reference papers. It can contain unsupported combinations (but these should be detected with function failure).

### 4.1 TEST3: Variable memory cost

The goal is to verify and compare real used memory according to the memory cost parameter. Test also compares impact to run time (large used memory accesses are expensive). Test increases memory cost according to Table 1 and measures used memory and run time. Other parameters are fixed: salt 16 bytes, input 32 bytes, output 32 bytes.

Candidate	Memory min	step	max	time cost
Argon	0	100	8000	0
Argon-AESNI	0	100	8000	0
Argon2d	0	100	8000	0
Argon2i	0	100	8000	0
battcrypt	0	1	16	0
Catena-Butterfly	0	1	24	1
Catena-Dragonfly	0	1	24	1
Lyra2	0	200	9000	1
Lyra2-SSE	0	200	9000	1
MAKWA	-	-	-	-
POMELO	0	1	17	0
POMELO-SSE	0	1	17	0
Parallel	-	-	-	-
Pufferfish	0	1	15	0
yescrypt	0	1	20	0
yescrypt-SSE	0	1	20	0

Table 1: Used parameters for the variable memory cost test.

Figure 1 (for x86-64) and Figure 2 (for PPC64) illustrate real measured data.

### 4.2 TEST4: Variable time cost

The goal is to verify and compare run time according to the time cost parameter. Test also compares impact to used memory (in theory there shouldn't be significant increase).

Test increases time cost according to Table 2 and measures used memory and run time. Other parameters are fixed: salt 16 bytes, input 32 bytes, output 32 bytes.

Candidate	Time min	step	max	memory cost
Argon	0	10	260	500
Argon-AESNI	0	10	260	500
Argon2d	0	10	250	500
Argon2i	0	10	260	500
battcrypt	0	1	20	5
Catena-Butterfly	0	5	100	14
Catena-Dragonfly	0	5	100	14
Lyra2	0	1000	20000	100
Lyra2-SSE	0	1000	20000	100
MAKWA	0	10000	300000	0
POMELO	0	1	26	0
POMELO-SSE	0	1	15	5
Parallel	0	1	15	5
Pufferfish	0	1	9	10
yescrypt	0	10	300	10
yescrypt-SSE	0	10	300	10

Table 2: Used parameters for the variable time cost test.

Figure 3 (for x86-64) and Figure 4 (for PPC64) illustrate real measured data.

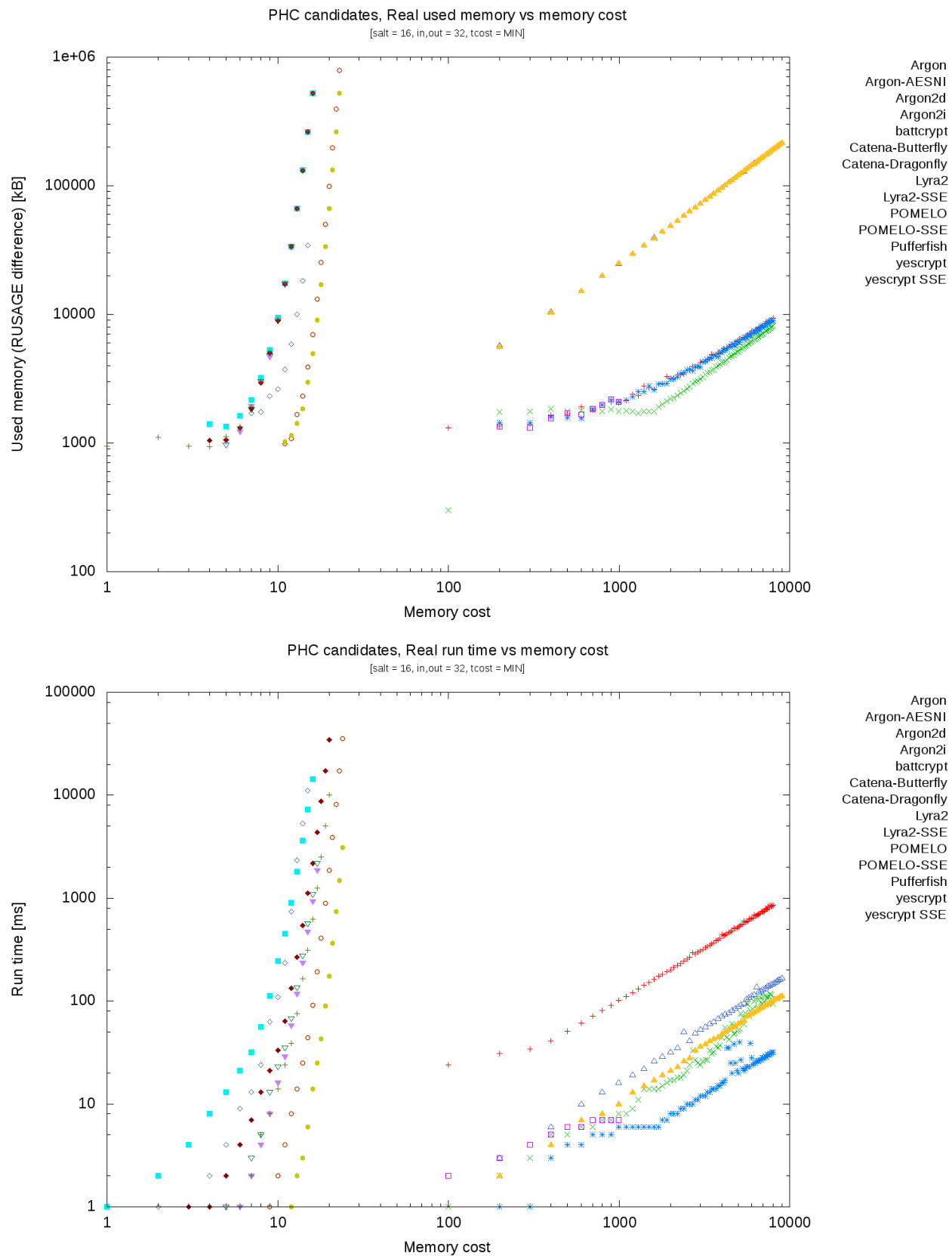


Figure 1: Real used memory and time dependence on memory cost parameter (x86-64).

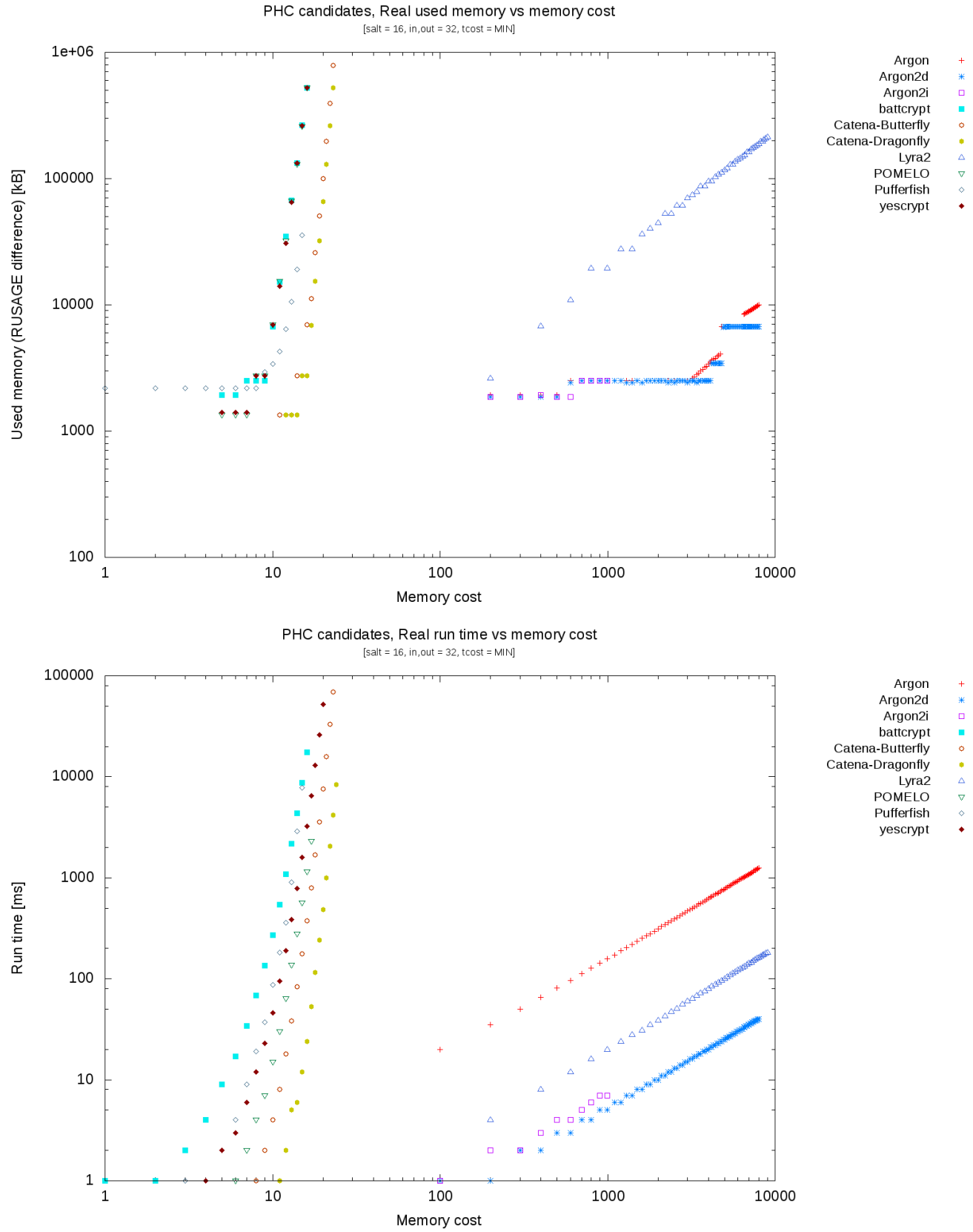


Figure 2: Real used memory and time dependence on memory cost parameter (PPC64).

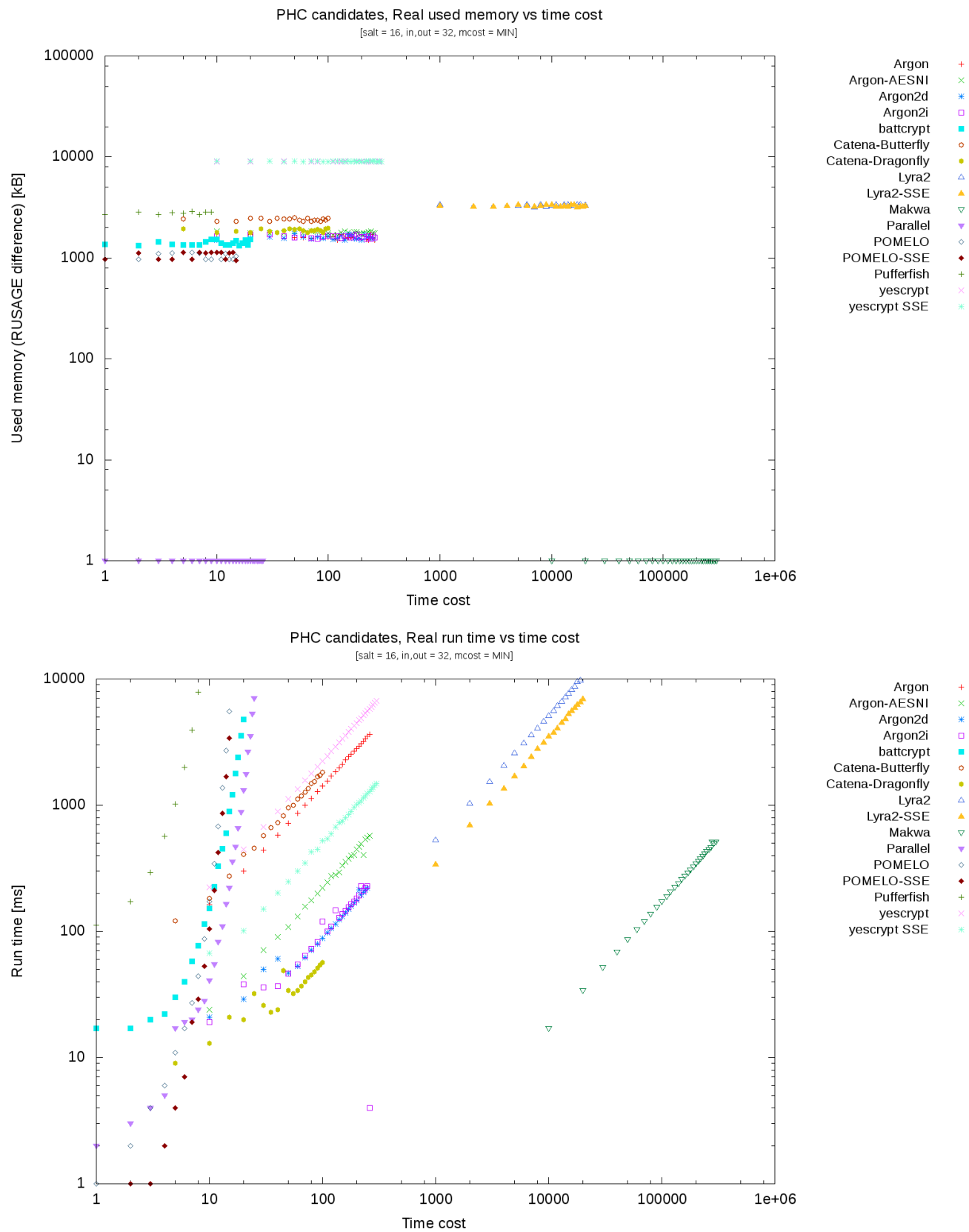


Figure 3: Real used memory and time dependence on time cost parameter (X86-64).

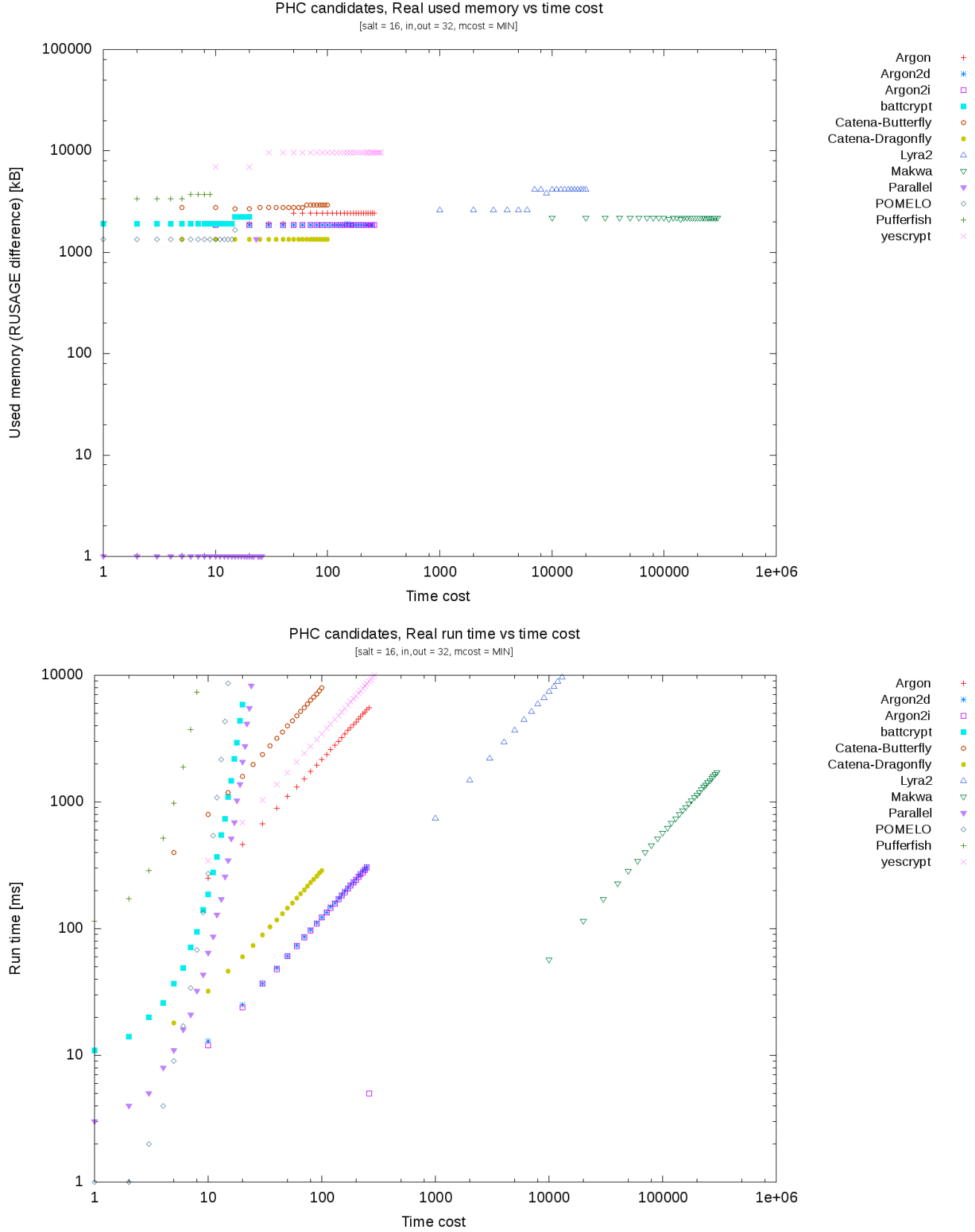


Figure 4: Real used memory and time dependence on time cost parameter (PPC64).



## 5 Password input and hash output length test

These tests should illustrate that run time is not dependent on input and output length.

### 5.1 TEST5: Impact of input length to run time (with fixed costs)

The input is random password increased from 1 to 300 bytes.

Other parameters are fixed to parameters in Table 3 (memory cost and time cost chosen to be minimal, just to provide run time around 50ms on the particular machine).

Parameters are: salt 16 bytes, output 32 bytes, input 1 - 300, step 1 byte.

Candidate	Memory cost	Time cost
Argon	0	2000
Argon-AESNI	0	1500
Argon2d	100	255
Argon2i	100	2000
battcrypt	0	17
Catena-Butterfly	10	60
Catena-Dragonfly	14	100
Lyra2	6	2000
Lyra2-SSE	6	3000
MAKWA	0	30000
POMELO	0	11
POMELO-SSE	0	13
Parallel	0	14
Pufferfish	0	12
yescrypt	10	1
yescrypt-SSE	10	10

Table 3: Used parameters for input and output length tests.

Figure 5 (for x86-64) and Figure 6 (for PPC64) illustrate real measured data.

### 5.2 TEST6: Impact of output lengths to run time (with fixed costs)

The output is increased from 1 to 300 bytes.

Other parameters are fixed to parameters in Table 3 (memory cost and time cost chosen to be minimal, just to provide run time around 50ms on the particular machine).

Parameters are: salt 16 bytes, input 32 bytes, output 1 - 300, step 1 byte.

Figure 7 (for x86-64) and Figure 8 (for PPC64) illustrate real measured data.

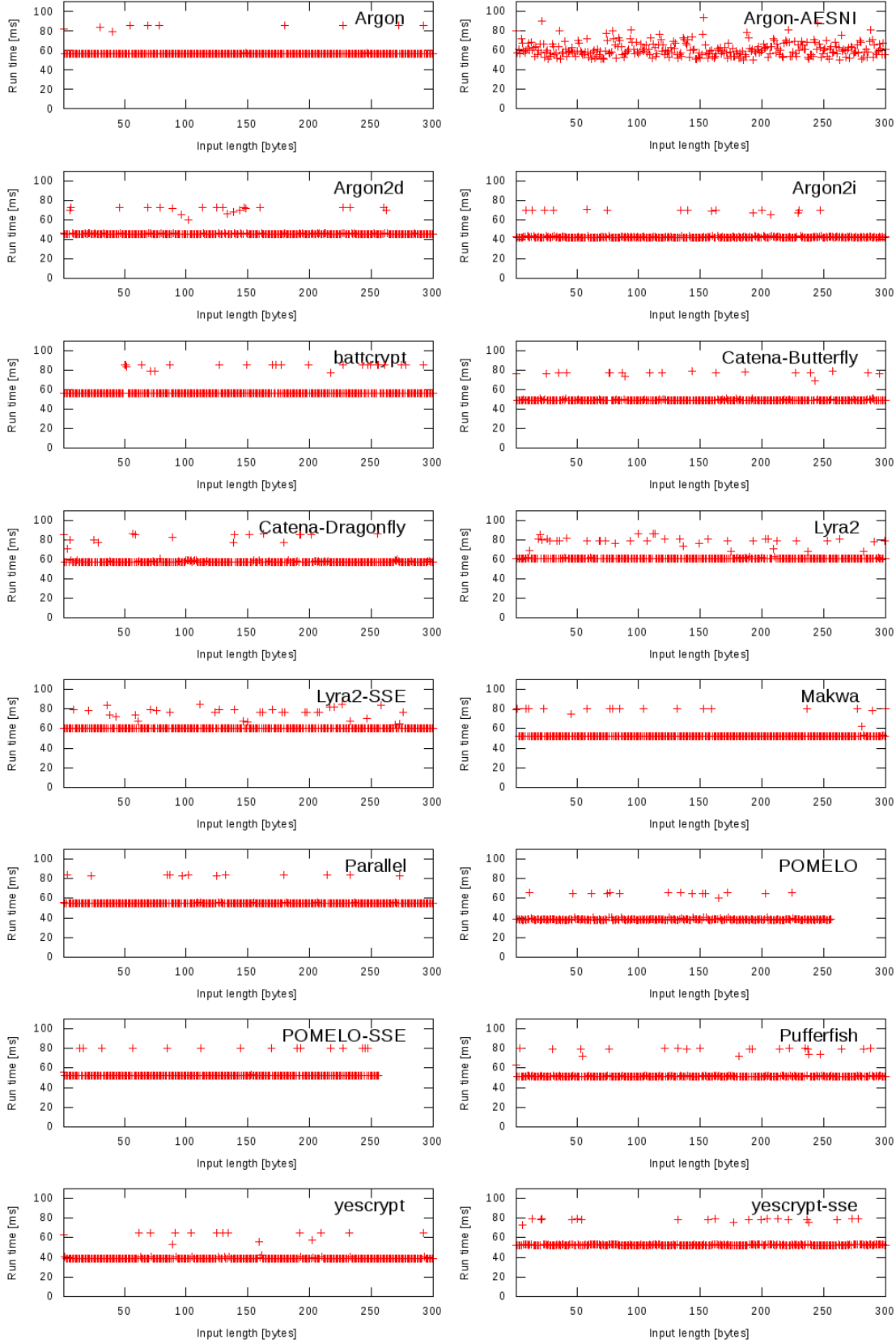


Figure 5: Input length test (X86-64).

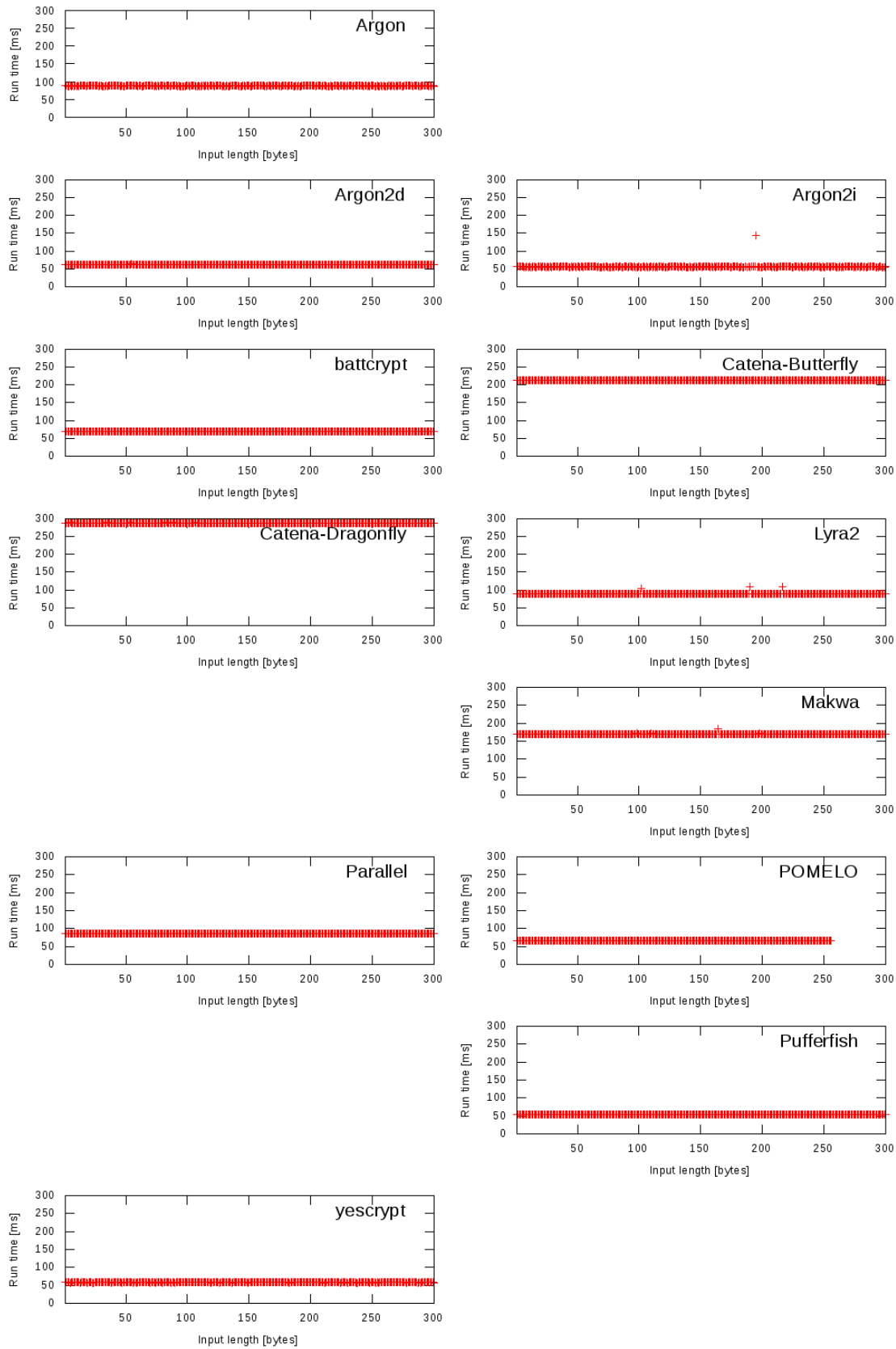


Figure 6: Input length test (PPC64).

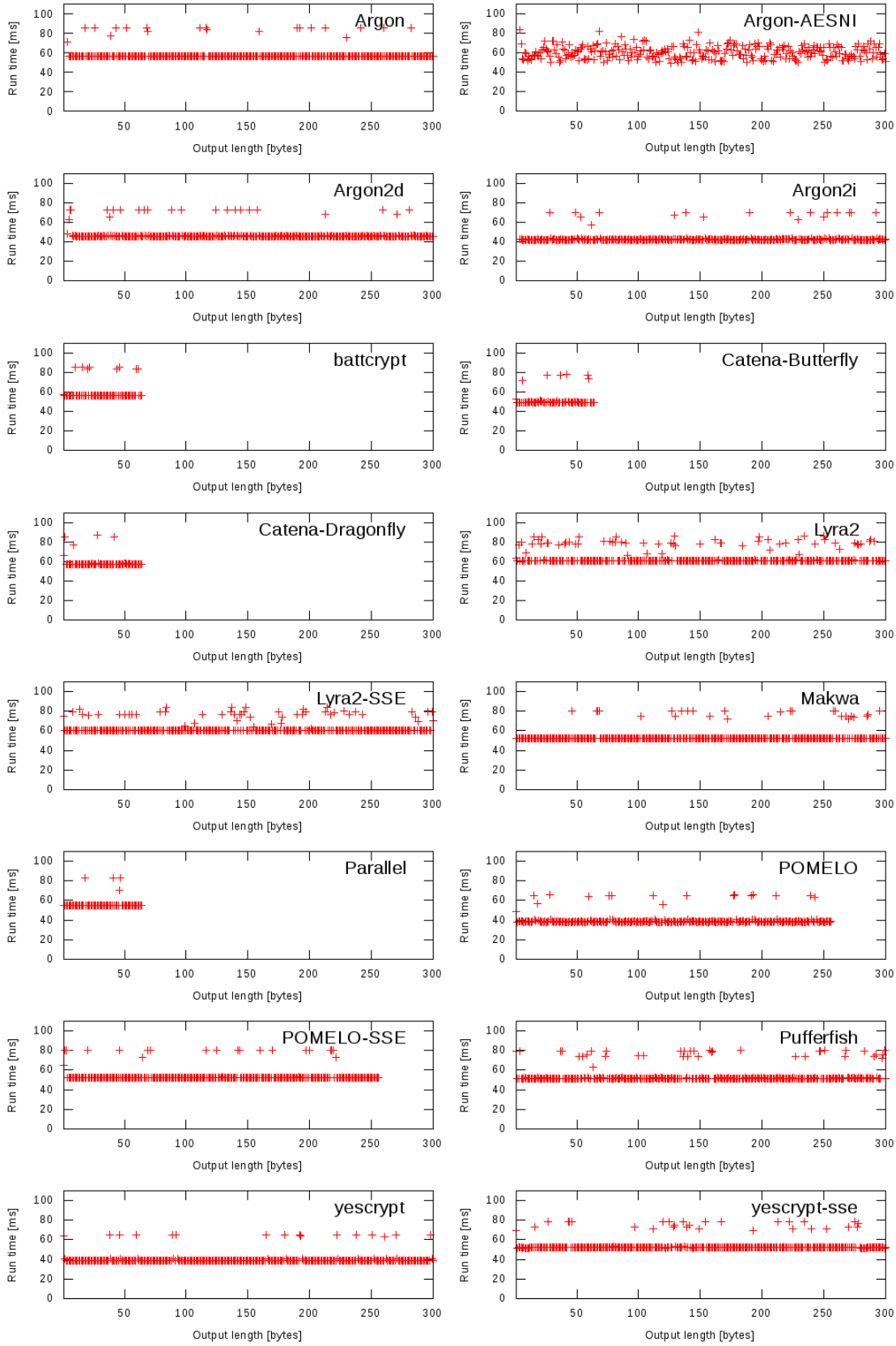


Figure 7: Output length test (X86-64).

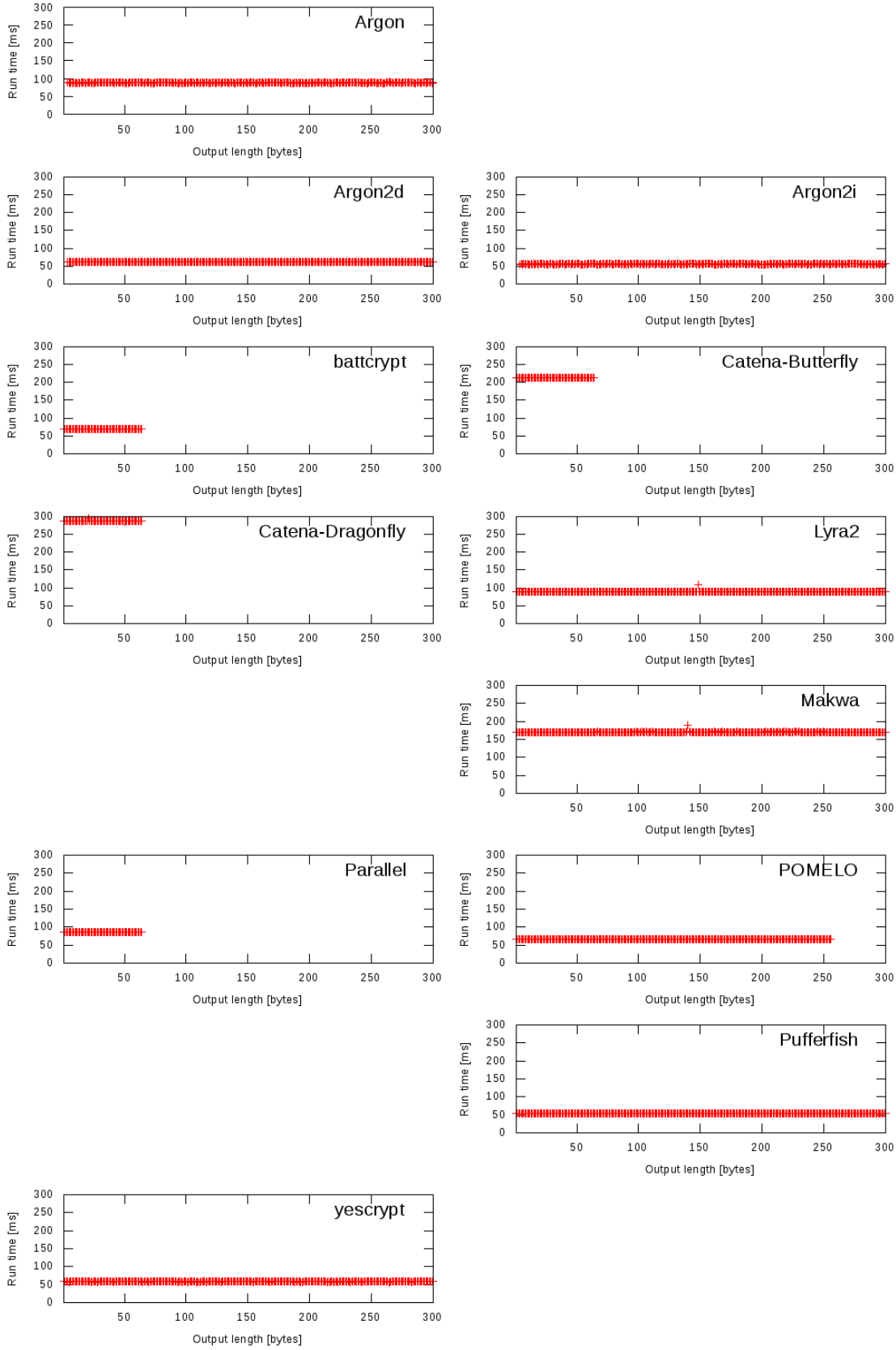


Figure 8: Output length test (PPC64).

## 6 TEST7: Real use case test for KDF mode

This test tries to answer the question if a function is able to provide usable parameters on the real test machine for the given set of memory and time use conditions.

The test limits memory to 1MB, 100MB or 1GB and also limits run-time (the first is minimum with the required memory limit, the second is approximate 1 second and the last one is approximate 20 seconds). The selection of one second simulates the default iteration time used in existing LUKS FDE system, 20 seconds represents limit which is usually yet acceptable for users when waiting for unlocking of a device.

Because attributes can be set in discrete steps only, measured time and memory for different candidates are slightly different and cannot be directly compared to each other. The Table 5 contains used parameters, the Table 4 then just conclusion if it is possible to provide particular tested case on this system.

(Note that Catena and battcrypt require code change for KDF mode and this test run in hashing mode only. The KDF modification mainly provides variable length output and because we need only 64 bytes (that equals internal block size) the modification can be ignored for this particular test case.)

Candidate	Memory	MIN	~1s	~20s
Argon	1MB	✓	✓	✓
	100MB	[1.9s] ✓	×	✓
	1GB	[18.4s] ✓	×	✓
Argon-AESNI	1MB	✓	✓	✓
	100MB	[1.9s] ✓	✓	✓
	1GB	[18.5s] ✓	×	✓
battcrypt	1MB	✓	✓	✓
	100MB	[3.7s] ✓	×	✓
	1GB	[28.9s] ✓	×	×
Catena Butterfly	1MB	✓	✓	×
	100MB	[1.9s] ✓	×	✓
	1GB	[36.6s] ✓	×	×
Lyra2	1MB	✓	✓	✓
	100MB	✓	✓	✓
	1GB	✓	✓	✓
Lyra2-SSE	1MB	✓	✓	✓
	100MB	✓	✓	✓
	1GB	✓	✓	✓
yescrypt	1MB	✓	✓	✓
	100MB	✓	✓	✓
	1GB	[4.3s] ✓	×	✓
yescrypt-SSE	1MB	✓	✓	✓
	100MB	✓	✓	✓
	1GB	[1.3s] ✓	×	✓

Table 4: Ability to cover real use-case limits

## 7 Conclusion

Provided tests illustrates limits and some issues with PHC candidates functions.

Unfortunately there are also implementation issues. As *TEST1* shows, some candidates are not prepared to run in big-endian environment without changes. These are technical problems which can be easily fixed later. *TEST3* and *TEST4* show that all candidates (except MAKWA and Parallel) are able to setup time and memory cost but some quite limits possible range of usable attributes.

The *TEST5* shows that functions can take at least 300 character long input (except POMELO which is limited to 255). *TEST6* is just illustration of the fact that some function need modifications if variable length output is requested.

The key-derivation test *TEST7* simulates one specific use case. Even here is visible that some candidates are not able to provide flexible limits.

Candidate	Memory	m_cost	min t_cost	~1s t_cost	~20s t_cost
Argon	1MB	1024	2	32	700
	100MB	10240	2	×	58
	1GB	102400	2	×	5
Argon-AESNI	1MB	1024	2	410	8200
	100MB	10240	2	5	103
	1GB	102400	2	×	9
battcrypt	1MB	6	0	17	26
	100MB	14	0	×	6
	1GB	17	0	×	×
Catena Butterfly	1MB	14	1	250	×
	100MB	20	1	×	n/a
	1GB	24	1	×	11
Lyra2	1MB	67	1	20000	×
	100MB	5000	1	35	400000
	1GB	43000	1	2	720
Lyra2-SSE	1MB	67	1	30000	81
	100MB	5000	1	45	600000
	1GB	43000	1	4	930
yescrypt	1MB	8	0	170	109
	100MB	14	0	3	3700
	1GB	17	0	×	55
yescrypt-SSE	1MB	8	0	900	7
	100MB	14	0	12	19000
	1GB	17	0	×	238
					27

Table 5: Used parameters for the run time example.

## References

- [1] Password hashing competition, 2014. <https://password-hashing.net/>.
- [2] Milan Broz. Benchmarks of the PHC candidates, November 2014. <http://htmlpreview.github.io/?https://github.com/mbroz/PHCTest/blob/master/output/index.html>.
- [3] Alex Biryukov and Dmitry Khovratovich. Argon and argon2, January 2015.
- [4] Steven Thomas. battcrypt (Blowfish All The Things), March 2014.
- [5] Christian Forler, Stefan Lucks, and Jakob Wenzel. The Catena Password-Scrambling Framework, January 2015.
- [6] Marcos A. Simplicio Jr., Leonardo C. Almeida, Ewerton R. Andrade, Paulo S. L. M. Barreto, and Marcos. The Lyra2 reference guide, January 2015.
- [7] Thomas Pornin. The MAKWA Password Hashing Function, March 2014.
- [8] Steven Thomas. Parallel, January 2015.
- [9] Hongjun Wu. POMELO: A Password Hashing Algorithm, January 2015.
- [10] Jeremi M. Gosney. The pufferfish password hashing scheme, March 2014.
- [11] Alexander Peslyak. yescrypt – a password hashing competition submission, January 2015.