eBASH: ECRYPT Benchmarking of All Submitted Hashes

http://bench.cr.yp.to
/ebash.html

D. J. Bernstein
University of Illinois at Chicago

Joint work with:
Tanja Lange
Technische Universiteit Eindhoven

European Union has funded NESSIE project (2000–2003), ECRYPT I network (2004–2008), ECRYPT II network (2008–2012).

NESSIE's performance evaluators tuned C implementations of many cryptographic systems, all supporting the same API; wrote a benchmarking toolkit; ran the toolkit on 25 computers.

Many specific performance results: e.g., 24 cycles/byte on P4 for 128-bit AES encryption.

I:
PT Benchmarking
Submitted Hashes

//bench.cr.yp.to
n.html

Bernstein
sity of Illinois at Chicago

work with:
Lange
sche Universiteit Eindhoven

European Union has funded
NESSIE project (2000–2003),
ECRYPT I network (2004–2008),
ECRYPT II network (2008–2012).

NESSIE's performance evaluators
tuned C implementations
of many cryptographic systems,
all supporting the same API;
wrote a benchmarking toolkit;
ran the toolkit on 25 computers.

Many specific performance results:
e.g., 24 cycles/byte on P4
for 128-bit AES encryption.

ECRYP
STVL,
include
STVL
ran eST

De Car
eSTRE

Stream
matchi
were co
publish
benchn

e.g. 18
third-p

marking
Hashes

cr.yp.to

ois at Chicago

ersiteit Eindhoven

European Union has funded
NESSIE project (2000–2003),
ECRYPT I network (2004–2008),
ECRYPT II network (2008–2012).

NESSIE's performance evaluators
tuned C implementations
of many cryptographic systems,
all supporting the same API;
wrote a benchmarking toolkit;
ran the toolkit on 25 computers.

Many specific performance results:
e.g., 24 cycles/byte on P4
for 128-bit AES encryption.

ECRYPT I had f
STVL, symmetri
included four wo
STVL WG 1, str
ran eSTREAM (

De Cannière *pub*
eSTREAM bench

Stream-cipher im
matching the be
were contributed
published, often
benchmarked on

e.g. 18 cycles/by
third-party asm

cago

dhoven

European Union has funded
NESSIE project (2000–2003),
ECRYPT I network (2004–2008),
ECRYPT II network (2008–2012).

NESSIE's performance evaluators
tuned C implementations
of many cryptographic systems,
all supporting the same API;
wrote a benchmarking toolkit;
ran the toolkit on 25 computers.

Many specific performance results:
e.g., 24 cycles/byte on P4
for 128-bit AES encryption.

ECRYPT I had five "virtua
STVL, symmetric-techniqu
included four working grou
STVL WG 1, stream-ciphe
ran eSTREAM (2004–200

De Cannière *published*
eSTREAM benchmarking t

Stream-cipher implementat
matching the benchmarkin
were contributed by design
published, often tuned;
benchmarked on many com

e.g. 18 cycles/byte on P4 f
third-party asm AES in too

European Union has funded
NESSIE project (2000–2003),
ECRYPT I network (2004–2008),
ECRYPT II network (2008–2012).

NESSIE's performance evaluators
tuned C implementations
of many cryptographic systems,
all supporting the same API;
wrote a benchmarking toolkit;
ran the toolkit on 25 computers.

Many specific performance results:
e.g., 24 cycles/byte on P4
for 128-bit AES encryption.

ECRYPT I had five "virtual labs."
STVL, symmetric-techniques lab,
included four working groups.
STVL WG 1, stream-cipher group,
ran eSTREAM (2004–2008).

De Cannière *published*
eSTREAM benchmarking toolkit.

Stream-cipher implementations
matching the benchmarking API
were contributed by designers,
published, often tuned;
benchmarked on many computers.

e.g. 18 cycles/byte on P4 for
third-party asm AES in toolkit.

ean Union has funded
E project (2000–2003),
PT I network (2004–2008),
PT II network (2008–2012).

E's performance evaluators
C implementations
y cryptographic systems,
porting the same API;
a benchmarking toolkit;
e toolkit on 25 computers.

specific performance results:
4 cycles/byte on P4
8-bit AES encryption.

ECRYPT I had five "virtual labs."
STVL, symmetric-techniques lab,
included four working groups.
STVL WG 1, stream-cipher group,
ran eSTREAM (2004–2008).

De Cannière *published*
eSTREAM benchmarking toolkit.

Stream-cipher implementations
matching the benchmarking API
were contributed by designers,
published, often tuned;
benchmarked on many computers.

e.g. 18 cycles/byte on P4 for
third-party asm AES in toolkit.

2006: V
Applica
Lab," s
("ECRY
of Asyr
measur
encrypt

Publish

Have w
49 pub
matchi
Benchr

has funded
(2000–2003),
ork (2004–2008),
vork (2008–2012).

mance evaluators
entations
raphic systems,
e same API;
arking toolkit;
n 25 computers.

rformance results:
yte on P4
encryption.

ECRYPT I had five “virtual labs.” STVL, symmetric-techniques lab, included four working groups. STVL WG 1, stream-cipher group, ran eSTREAM (2004–2008).

De Cannière *published* eSTREAM benchmarking toolkit.

Stream-cipher implementations matching the benchmarking API were contributed by designers, *published*, often tuned; benchmarked on many computers.

e.g. 18 cycles/byte on P4 for third-party asm AES in toolkit.

2006: VAMPIRE
Application and
Lab,” started eB
(“ECRYPT Benc
of Asymmetric S
measuring efficie
encryption, signa

Published a new

Have written, co
49 public-key im
matching the be
Benchmarked on

d
3),
-2008),
–2012).

luators

tems,
PI;
kit;
puters.

results:

.

ECRYPT I had five “virtual labs.” STVL, symmetric-techniques lab, included four working groups. STVL WG 1, stream-cipher group, ran eSTREAM (2004–2008).

De Cannière *published* eSTREAM benchmarking toolkit.

Stream-cipher implementations matching the benchmarking API were contributed by designers, *published*, often tuned; benchmarked on many computers.

e.g. 18 cycles/byte on P4 for third-party asm AES in toolkit.

2006: VAMPIRE, “Virtual
Application and Implement
Lab,” started eBATS
(“ECRYPT Benchmarking
of Asymmetric Systems”),
measuring efficiency of pub
encryption, signatures, DH

Published a new toolkit.

Have written, collected, pu
49 public-key implementati
matching the benchmarkin
Benchmarked on many con

ECRYPT I had five “virtual labs.” STVL, symmetric-techniques lab, included four working groups. STVL WG 1, stream-cipher group, ran eSTREAM (2004–2008).

De Cannière *published* eSTREAM benchmarking toolkit.

Stream-cipher implementations matching the benchmarking API were contributed by designers, *published*, often tuned; benchmarked on many computers.

e.g. 18 cycles/byte on P4 for third-party asm AES in toolkit.

2006: VAMPIRE, “Virtual Application and Implementation Lab,” started eBATS (“ECRYPT Benchmarking of Asymmetric Systems”), measuring efficiency of public-key encryption, signatures, DH.

Published a new toolkit.

Have written, collected, published 49 public-key implementations matching the benchmarking API. Benchmarked on many computers.

PT I had five “virtual labs.”
symmetric-techniques lab,
ed four working groups.
WG 1, stream-cipher group,
TREAM (2004–2008).

nnière *published*
AM benchmarking toolkit.

n-cipher implementations
ng the benchmarking API
ontributed by designers,
ed, often tuned;
marked on many computers.

cycles/byte on P4 for
arty asm AES in toolkit.

2006: VAMPIRE, “Virtual Application and Implementation Lab,” started eBATS (“ECRYPT Benchmarking of Asymmetric Systems”), measuring efficiency of public-key encryption, signatures, DH.

Published a new toolkit.

Have written, collected, published 49 public-key implementations matching the benchmarking API. Benchmarked on many computers.

2008: V
(“ECRY
of Stre
post-eS

VAMP
(“ECRY
of All S

eBACS
of Cryp
include
Contin

New to
with C
AES no

ive "virtual labs."
c-techniques lab,
rking groups.
eam-cipher group,
2004–2008).

blished
hmarking toolkit.

nplementations
nchmarking API
by designers,
tuned;
many computers.

te on P4 for
AES in toolkit.

2006: VAMPIRE, "Virtual Application and Implementation Lab," started eBATS ("ECRYPT Benchmarking of Asymmetric Systems"), measuring efficiency of public-key encryption, signatures, DH.

Published a new toolkit.

Have written, collected, published 49 public-key implementations matching the benchmarking API. Benchmarked on many computers.

2008: VAMPIRE
("ECRYPT Benc
of Stream Cipher
post-eSTREAM

VAMPIRE also st
("ECRYPT Benc
of All Submitted

eBACS ("ECRYP
of Cryptographic
includes eBATS,
Continues under

New toolkit, API
with CACE librar
AES now 14 cycl

al labs.”
es lab,
ps.
r group,
3).

toolkit.

tions
g API
ers,

nputers.

for
lkit.

2006: VAMPIRE, “Virtual Application and Implementation Lab,” started eBATS (“ECRYPT Benchmarking of Asymmetric Systems”), measuring efficiency of public-key encryption, signatures, DH.

Published a new toolkit.

Have written, collected, published 49 public-key implementations matching the benchmarking API. Benchmarked on many computers.

2008: VAMPIRE started e
(“ECRYPT Benchmarking
of Stream Ciphers”) for
post-eSTREAM benchmark

VAMPIRE also started eB
(“ECRYPT Benchmarking
of All Submitted Hashes”)

eBACS (“ECRYPT Bench
of Cryptographic Systems”
includes eBATS, eBASH, e
Continues under ECRYPT

New toolkit, API; coordina
with CACE library (NaCl).
AES now 14 cycles/byte o

2006: VAMPIRE, “Virtual Application and Implementation Lab,” started eBATS (“ECRYPT Benchmarking of Asymmetric Systems”), measuring efficiency of public-key encryption, signatures, DH.

Published a new toolkit.

Have written, collected, published 49 public-key implementations matching the benchmarking API. Benchmarked on many computers.

2008: VAMPIRE started eBASC (“ECRYPT Benchmarking of Stream Ciphers”) for post-eSTREAM benchmarks.

VAMPIRE also started eBASH (“ECRYPT Benchmarking of All Submitted Hashes”).

eBACS (“ECRYPT Benchmarking of Cryptographic Systems”) includes eBATS, eBASH, eBASC. Continues under ECRYPT II.

New toolkit, API; coordinated with CACE library (NaCl). AES now 14 cycles/byte on P4.

VAMPIRE, “Virtual
ation and Implementation
started eBATS
YPT Benchmarking
mmetric Systems”),
ring efficiency of public-key
tion, signatures, DH.

ned a new toolkit.

vritten, collected, published
lic-key implementations
ng the benchmarking API.
marked on many computers.

2008: VAMPIRE started eBASC (“ECRYPT Benchmarking of Stream Ciphers”) for post-eSTREAM benchmarks.

VAMPIRE also started eBASH (“ECRYPT Benchmarking of All Submitted Hashes”).

eBACS (“ECRYPT Benchmarking of Cryptographic Systems”) includes eBATS, eBASH, eBASC. Continues under ECRYPT II.

New toolkit, API; coordinated with CACE library (NaCl). AES now 14 cycles/byte on P4.

eBASH

eBASH
77 imp
38 hash

http:/
/resul
already
measur
101 ma

Each ir
recomp
with va
to iden
for imp

, “Virtual
Implementation
ATS
chmarking
ystems”),
ncy of public-key
tures, DH.

toolkit.

llected, published
plementations
nchmarking API.
many computers.

2008: VAMPIRE started eBASC (“ECRYPT Benchmarking of Stream Ciphers”) for post-eSTREAM benchmarks.

VAMPIRE also started eBASH (“ECRYPT Benchmarking of All Submitted Hashes”).

eBACS (“ECRYPT Benchmarking of Cryptographic Systems”) includes eBATS, eBASH, eBASC. Continues under ECRYPT II.

New toolkit, API; coordinated with CACE library (NaCl). AES now 14 cycles/byte on P4.

eBASH → public

eBASH has alrea
77 implementatio
38 hash function

http://bench.c
/results-hash
already shows
measurements on
101 machine-AB

Each implementa
recompiled 1226
with various com
to identify best w
for implementatio

tation

lic-key

.

blished

ions

g API.

mputers.

2008: VAMPIRE started eBASC (“ECRYPT Benchmarking of Stream Ciphers”) for post-eSTREAM benchmarks.

VAMPIRE also started eBASH (“ECRYPT Benchmarking of All Submitted Hashes”).

eBACS (“ECRYPT Benchmarking of Cryptographic Systems”) includes eBATS, eBASH, eBASC. Continues under ECRYPT II.

New toolkit, API; coordinated with CACE library (NaCl). AES now 14 cycles/byte on P4.

eBASH → public

eBASH has already collect

77 implementations of

38 hash functions in 18 fa

http://bench.cr.yp.to

/results-hash.html

already shows

measurements on 71 mach

101 machine-ABI combina

Each implementation is

recompiled 1226 times

with various compiler optio

to identify best working op

for implementation, machi

2008: VAMPIRE started eBASC ("ECRYPT Benchmarking of Stream Ciphers") for post-eSTREAM benchmarks.

VAMPIRE also started eBASH ("ECRYPT Benchmarking of All Submitted Hashes").

eBACS ("ECRYPT Benchmarking of Cryptographic Systems") includes eBATS, eBASH, eBASC. Continues under ECRYPT II.

New toolkit, API; coordinated with CACE library (NaCl). AES now 14 cycles/byte on P4.

eBASH → public

eBASH has already collected 77 implementations of 38 hash functions in 18 families.

`http://bench.cr.yp.to/results-hash.html` already shows measurements on 71 machines; 101 machine-ABI combinations.

Each implementation is recompiled 1226 times with various compiler options to identify best working option for implementation, machine.

VAMPIRE started eBASC
YPT Benchmarking
am Ciphers”) for
STREAM benchmarks.

IRE also started eBASH
YPT Benchmarking
Submitted Hashes”).

s (“ECRYPT Benchmarking
ptographic Systems”)
es eBATS, eBASH, eBASC.
ues under ECRYPT II.

olkit, API; coordinated
ACE library (NaCl).
ow 14 cycles/byte on P4.

eBASH → public

eBASH has already collected 77 implementations of 38 hash functions in 18 families.

http://bench.cr.yp.to
/results-hash.html
already shows measurements on 71 machines; 101 machine-ABI combinations.

Each implementation is recompiled 1226 times with various compiler options to identify best working option for implementation, machine.

e.g. 15
Duo 6f

25%
2.83
4.46
5.29
7.08
8.29
8.39
9.59
9.67
11.29
11.47
12.08
12.05
14.83

. started eBASC
chmarking
rs") for
benchmarks.

tarted eBASH
chmarking
Hashes").

PT Benchmarking
Systems")
eBASH, eBASC.
ECRYPT II.

; coordinated
ry (NaCl).
es/byte on P4.

eBASH → public

eBASH has already collected
77 implementations of
38 hash functions in 18 families.

http://bench.cr.yp.to
/results-hash.html
already shows
measurements on 71 machines;
101 machine-ABI combinations.

Each implementation is
recompiled 1226 times
with various compiler options
to identify best working option
for implementation, machine.

e.g. 1536 bytes,
Duo 6f6, 2137M

25%	50%	75
2.83	2.83	2.8
4.46	4.46	4.4
5.29	5.30	5.3
7.08	7.08	7.0
8.29	8.30	8.3
8.39	8.39	8.4
9.59	9.59	9.6
9.67	9.76	9.7
11.29	11.30	11.3
11.47	11.49	11.5
12.08	12.08	12.0
12.05	12.09	12.0
14.83	14.83	14.8

BASC

ks.

ASH

-

marking

)

eBASC.

II.

ted

n P4.

eBASH → public

eBASH has already collected
77 implementations of
38 hash functions in 18 families.

http://bench.cr.yp.to
/results-hash.html
already shows
measurements on 71 machines;
101 machine-ABI combinations.

Each implementation is
recompiled 1226 times
with various compiler options
to identify best working option
for implementation, machine.

e.g. 1536 bytes, katana (C
Duo 6f6, 2137MHz), 64-bi

25%	50%	75%	hash
2.83	2.83	2.83	edonr5
4.46	4.46	4.46	bmw51
5.29	5.30	5.38	edonr2
7.08	7.08	7.08	skein51
8.29	8.30	8.30	sha1
8.39	8.39	8.47	bmw25
9.59	9.59	9.60	cubeha
9.67	9.76	9.76	shabal5
11.29	11.30	11.30	keccak
11.47	11.49	11.54	simd25
12.08	12.08	12.08	blake64
12.05	12.09	12.09	blake32
14.83	14.83	14.85	sha512
			etc.

eBASH → public

eBASH has already collected 77 implementations of 38 hash functions in 18 families.

http://bench.cr.yp.to /results-hash.html already shows measurements on 71 machines; 101 machine-ABI combinations.

Each implementation is recompiled 1226 times with various compiler options to identify best working option for implementation, machine.

e.g. 1536 bytes, katana (Core 2 Duo 6f6, 2137MHz), 64-bit ABI:

25%	50%	75%	hash
2.83	2.83	2.83	edonr512
4.46	4.46	4.46	bmw512
5.29	5.30	5.38	edonr256
7.08	7.08	7.08	skein512
8.29	8.30	8.30	sha1
8.39	8.39	8.47	bmw256
9.59	9.59	9.60	cubehash832
9.67	9.76	9.76	shabal512
11.29	11.30	11.30	keccakr1024c576
11.47	11.49	11.54	simd256
12.08	12.08	12.08	blake64
12.05	12.09	12.09	blake32
14.83	14.83	14.85	sha512
			etc.

l → public

l has already collected
lementations of
h functions in 18 families.

//bench.cr.yp.to
lts-hash.html
y shows
rements on 71 machines;
achine-ABI combinations.

mplementation is
oiled 1226 times
arious compiler options
tify best working option
plementation, machine.

e.g. 1536 bytes, `katana` (Core 2 Duo 6f6, 2137MHz), 64-bit ABI:

25%	50%	75%	hash
2.83	2.83	2.83	edonr512
4.46	4.46	4.46	bmw512
5.29	5.30	5.38	edonr256
7.08	7.08	7.08	skein512
8.29	8.30	8.30	sha1
8.39	8.39	8.47	bmw256
9.59	9.59	9.60	cubehash832
9.67	9.76	9.76	shabal512
11.29	11.30	11.30	keccakr1024c576
11.47	11.49	11.54	simd256
12.08	12.08	12.08	blake64
12.05	12.09	12.09	blake32
14.83	14.83	14.85	sha512
			etc.

Tables
of cycl
8-byte
64-byt
576-by
1536-b
4096-b
(extrap

Actuall
e.g. Re
e.g. Gr
0-byte
2-byte
4-byte
. . ., 20

c

dy collected
ons of
s in 18 families.

cr.yp.to
.html

n 71 machines;
l combinations.

ation is
times
piler options
working option
on, machine.

e.g. 1536 bytes, `katana` (Core 2 Duo 6f6, 2137MHz), 64-bit ABI:

25%	50%	75%	hash
2.83	2.83	2.83	edonr512
4.46	4.46	4.46	bmw512
5.29	5.30	5.38	edonr256
7.08	7.08	7.08	skein512
8.29	8.30	8.30	sha1
8.39	8.39	8.47	bmw256
9.59	9.59	9.60	cubehash832
9.67	9.76	9.76	shabal512
11.29	11.30	11.30	keccakr1024c576
11.47	11.49	11.54	simd256
12.08	12.08	12.08	blake64
12.05	12.09	12.09	blake32
14.83	14.83	14.85	sha512
			etc.

Tables show med
of cycles/byte to
8-byte message,
64-byte message,
576-byte messag
1536-byte messa
4096-byte messa
(extrapolated) lo

Actually have mu
e.g. Reports show
e.g. Graphs show
0-byte message,
2-byte message,
4-byte message,
. . ., 2048-byte m

ed

milies.

ines;

tions.

ons

otion

ne.

e.g. 1536 bytes, `katana` (Core 2 Duo 6f6, 2137MHz), 64-bit ABI:

25%	50%	75%	hash
2.83	2.83	2.83	edonr512
4.46	4.46	4.46	bmw512
5.29	5.30	5.38	edonr256
7.08	7.08	7.08	skein512
8.29	8.30	8.30	sha1
8.39	8.39	8.47	bmw256
9.59	9.59	9.60	cubehash832
9.67	9.76	9.76	shabal512
11.29	11.30	11.30	keccakr1024c576
11.47	11.49	11.54	simd256
12.08	12.08	12.08	blake64
12.05	12.09	12.09	blake32
14.83	14.83	14.85	sha512
			etc.

Tables show medians, quar
of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long messag

Actually have much more d
e.g. Reports show best opt
e.g. Graphs show medians
0-byte message, 1-byte me
2-byte message, 3-byte me
4-byte message, 5-byte me
. . ., 2048-byte message.

e.g. 1536 bytes, `katana` (Core 2 Duo 6f6, 2137MHz), 64-bit ABI:

25%	50%	75%	hash
2.83	2.83	2.83	edonr512
4.46	4.46	4.46	bmw512
5.29	5.30	5.38	edonr256
7.08	7.08	7.08	skein512
8.29	8.30	8.30	sha1
8.39	8.39	8.47	bmw256
9.59	9.59	9.60	cubehash832
9.67	9.76	9.76	shabal512
11.29	11.30	11.30	keccakr1024c576
11.47	11.49	11.54	simd256
12.08	12.08	12.08	blake64
12.05	12.09	12.09	blake32
14.83	14.83	14.85	sha512
			etc.

Tables show medians, quartiles of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long message.

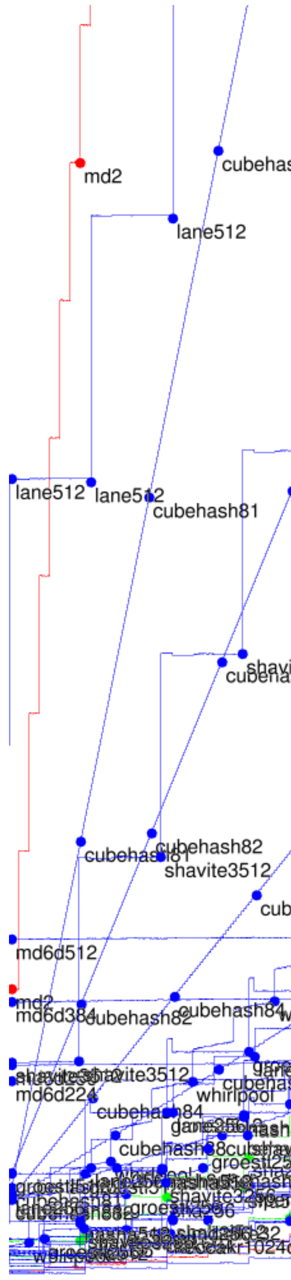
Actually have much more data.
e.g. Reports show best options.
e.g. Graphs show medians for 0-byte message, 1-byte message, 2-byte message, 3-byte message, 4-byte message, 5-byte message, . . ., 2048-byte message.

36 bytes, katana (Core 2
6, 2137MHz), 64-bit ABI:

50%	75%	hash
2.83	2.83	edonr512
4.46	4.46	bmw512
5.30	5.38	edonr256
7.08	7.08	skein512
8.30	8.30	sha1
8.39	8.47	bmw256
9.59	9.60	cubehash832
9.76	9.76	shabal512
11.30	11.30	keccakr1024c576
11.49	11.54	simd256
12.08	12.08	blake64
12.09	12.09	blake32
14.83	14.85	sha512
		etc.

Tables show medians, quartiles
of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long message.

Actually have much more data.
e.g. Reports show best options.
e.g. Graphs show medians for
0-byte message, 1-byte message,
2-byte message, 3-byte message,
4-byte message, 5-byte message,
. . ., 2048-byte message.

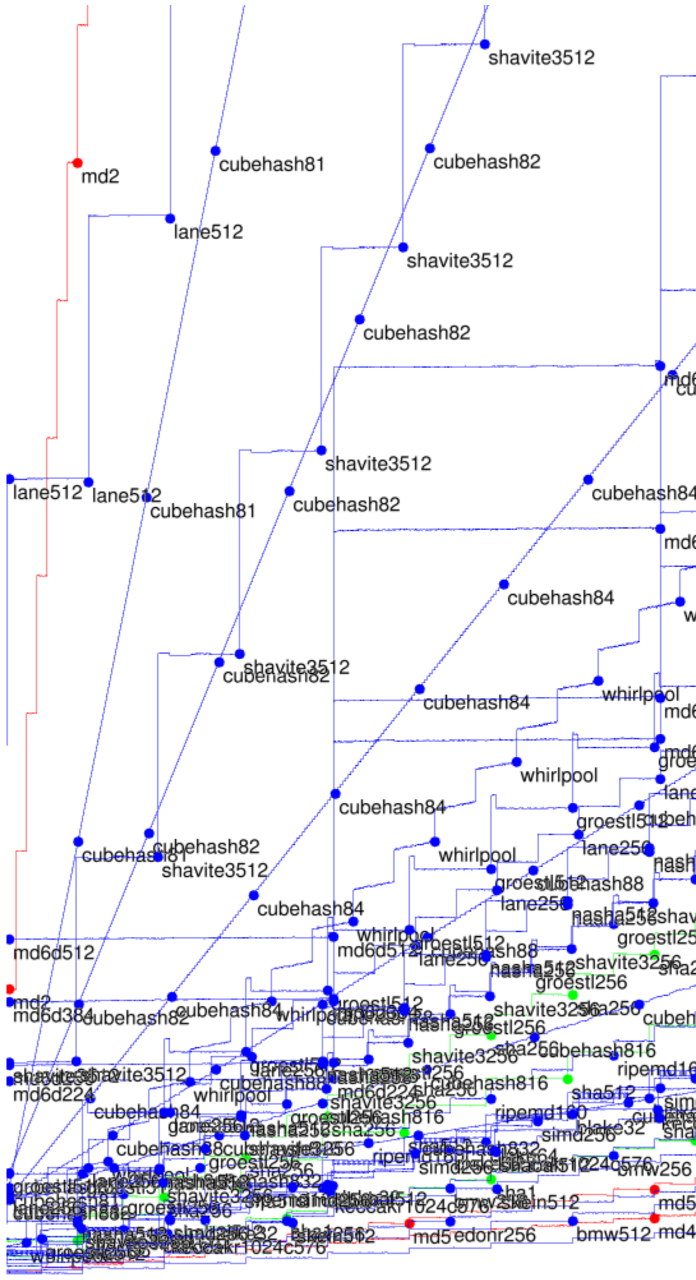


katana (Core 2
Hz), 64-bit ABI:

%	hash
33	edonr512
46	bmw512
38	edonr256
08	skein512
30	sha1
47	bmw256
60	cubehash832
76	shabal512
30	keccakr1024c576
54	simd256
08	blake64
09	blake32
35	sha512
	etc.

Tables show medians, quartiles of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long message.

Actually have much more data.
e.g. Reports show best options.
e.g. Graphs show medians for
0-byte message, 1-byte message,
2-byte message, 3-byte message,
4-byte message, 5-byte message,
. . ., 2048-byte message.



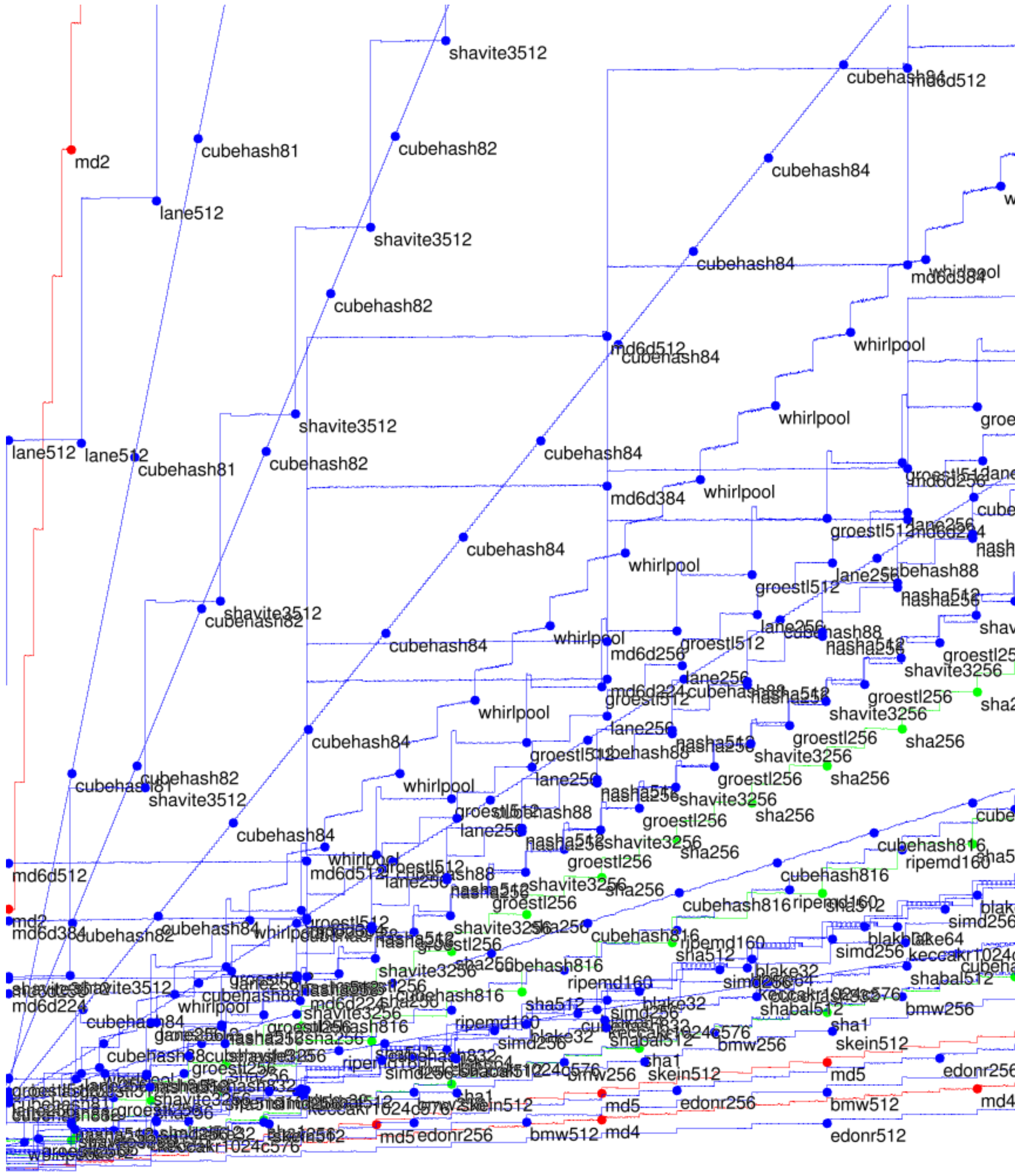
Core 2
t ABI:

12
2
56
2

6
sh832
512
r1024c576
6
4
2

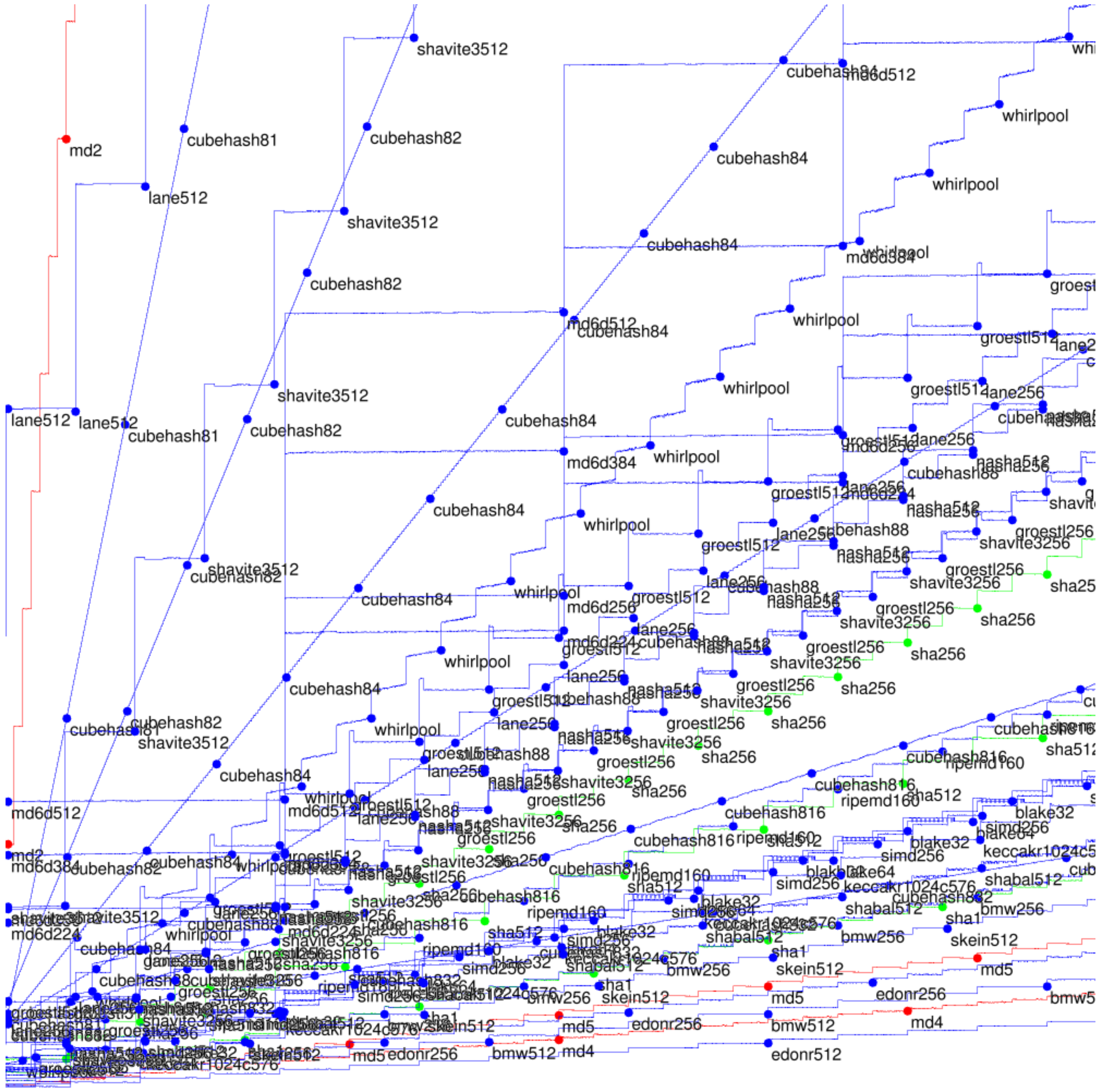
Tables show medians, quartiles of cycles/byte to hash 8-byte message, 64-byte message, 576-byte message, 1536-byte message, 4096-byte message, (extrapolated) long message.

Actually have much more data. e.g. Reports show best options. e.g. Graphs show medians for 0-byte message, 1-byte message, 2-byte message, 3-byte message, 4-byte message, 5-byte message, . . ., 2048-byte message.



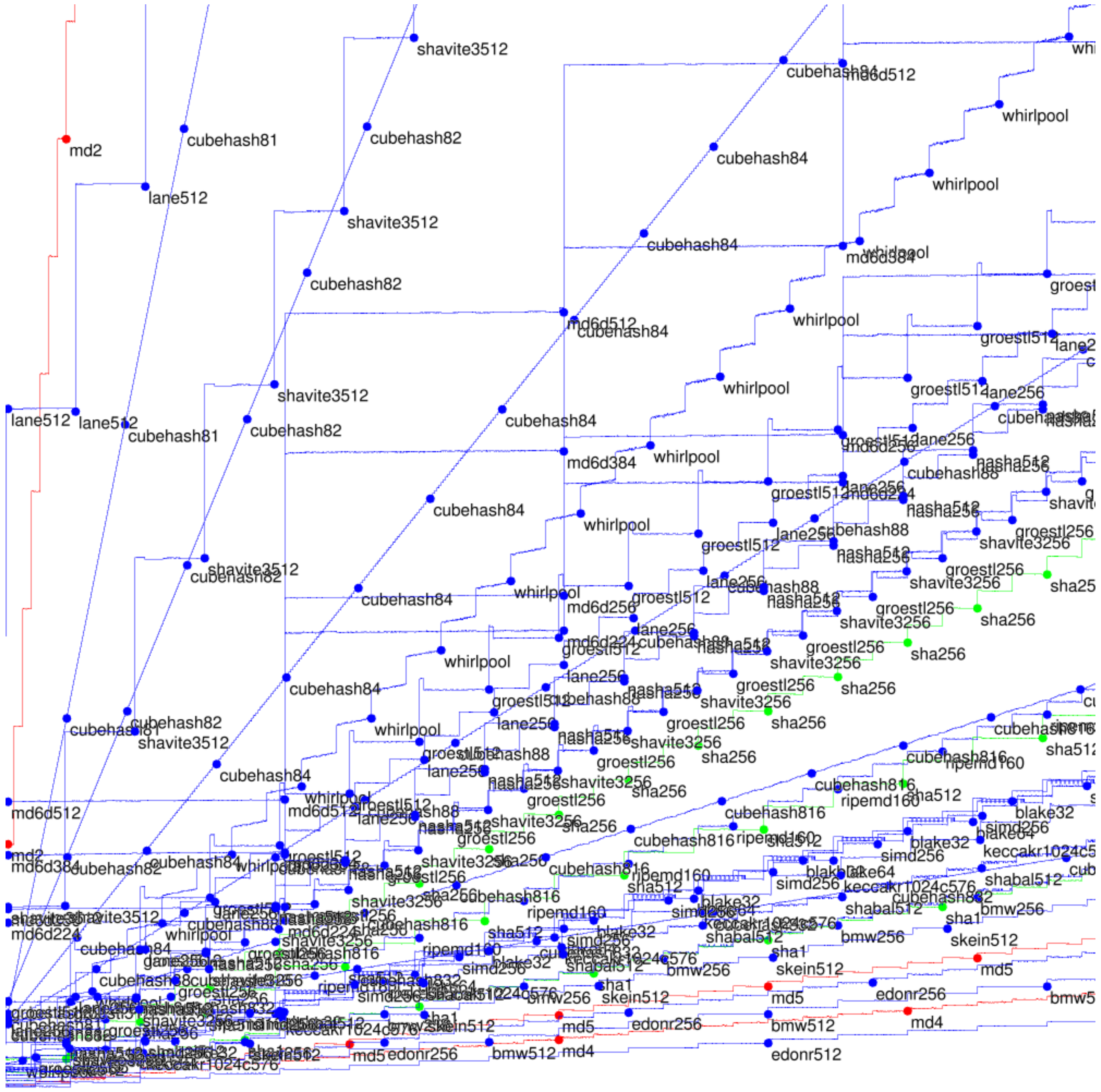
Tables show medians, quartiles of cycles/byte to hash 8-byte message, 64-byte message, 576-byte message, 1536-byte message, 4096-byte message, (extrapolated) long message.

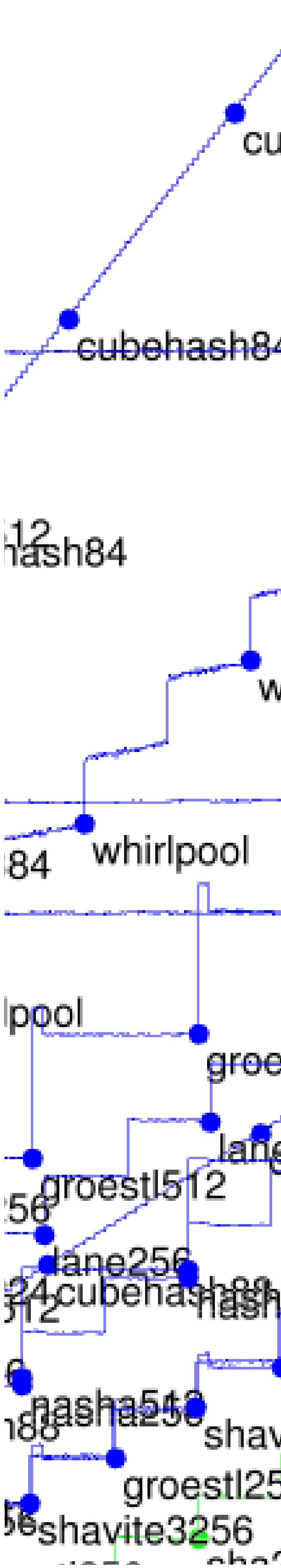
Actually have much more data. e.g. Reports show best options. e.g. Graphs show medians for 0-byte message, 1-byte message, 2-byte message, 3-byte message, 4-byte message, 5-byte message, . . ., 2048-byte message.



show medians, quartiles
es/byte to hash
message,
e message,
te message,
yte message,
yte message,
olated) long message.

ly have much more data.
eports show best options.
aphs show medians for
message, 1-byte message,
message, 3-byte message,
message, 5-byte message,
48-byte message.

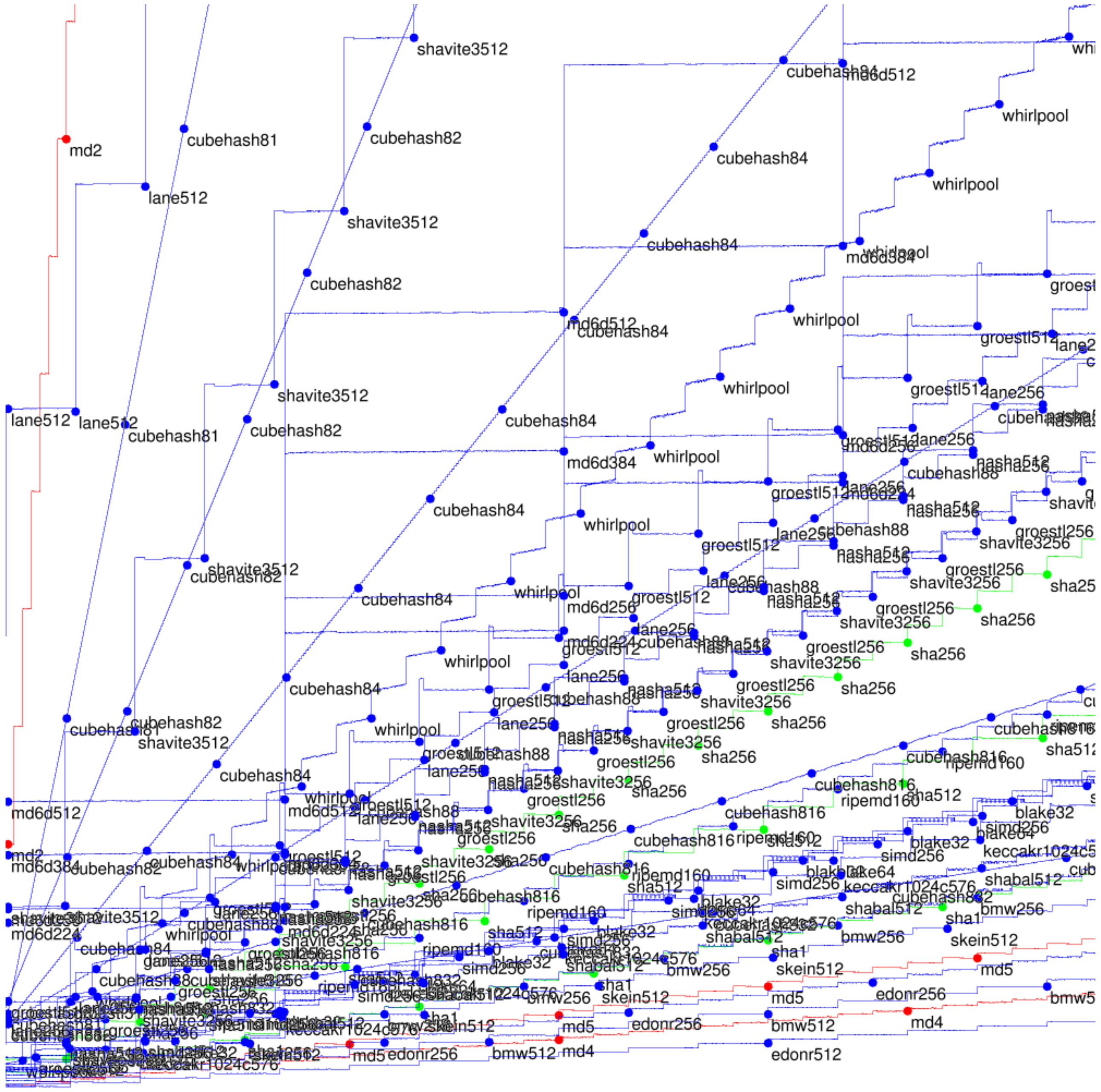


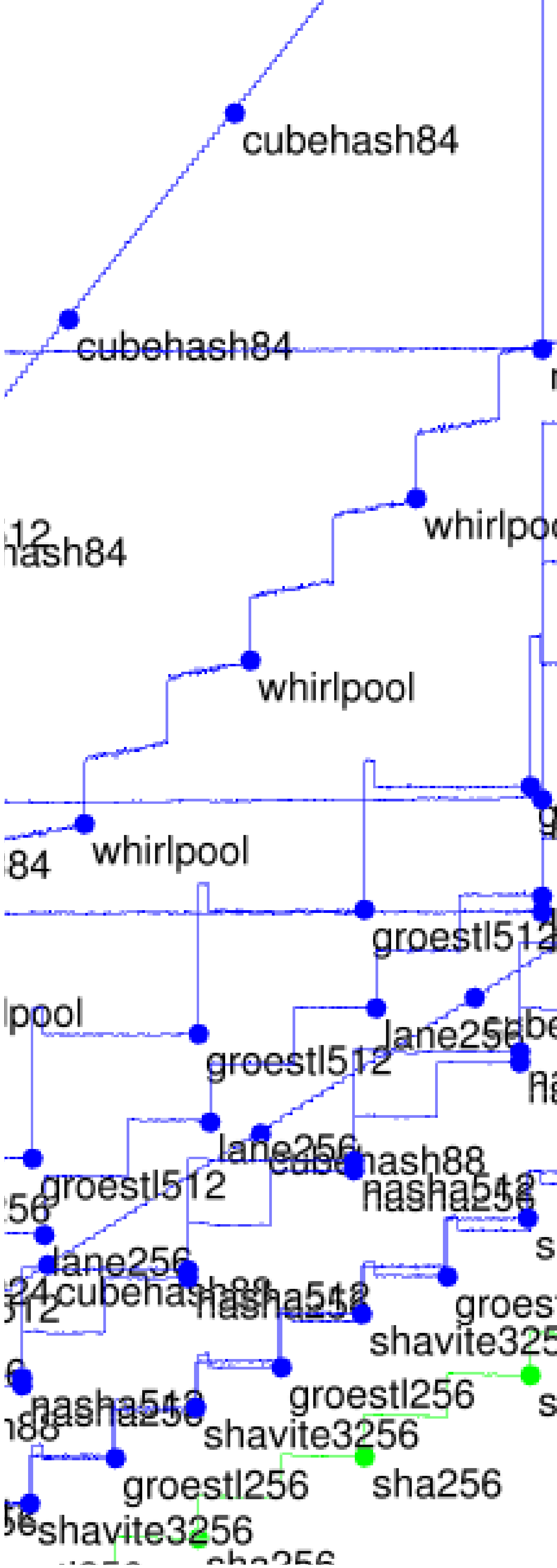


lians, quartiles
hash

,
e,
ge,
ge,
ng message.

uch more data.
w best options.
medians for
1-byte message,
3-byte message,
5-byte message,
essage.





rtiles

ge.

data.

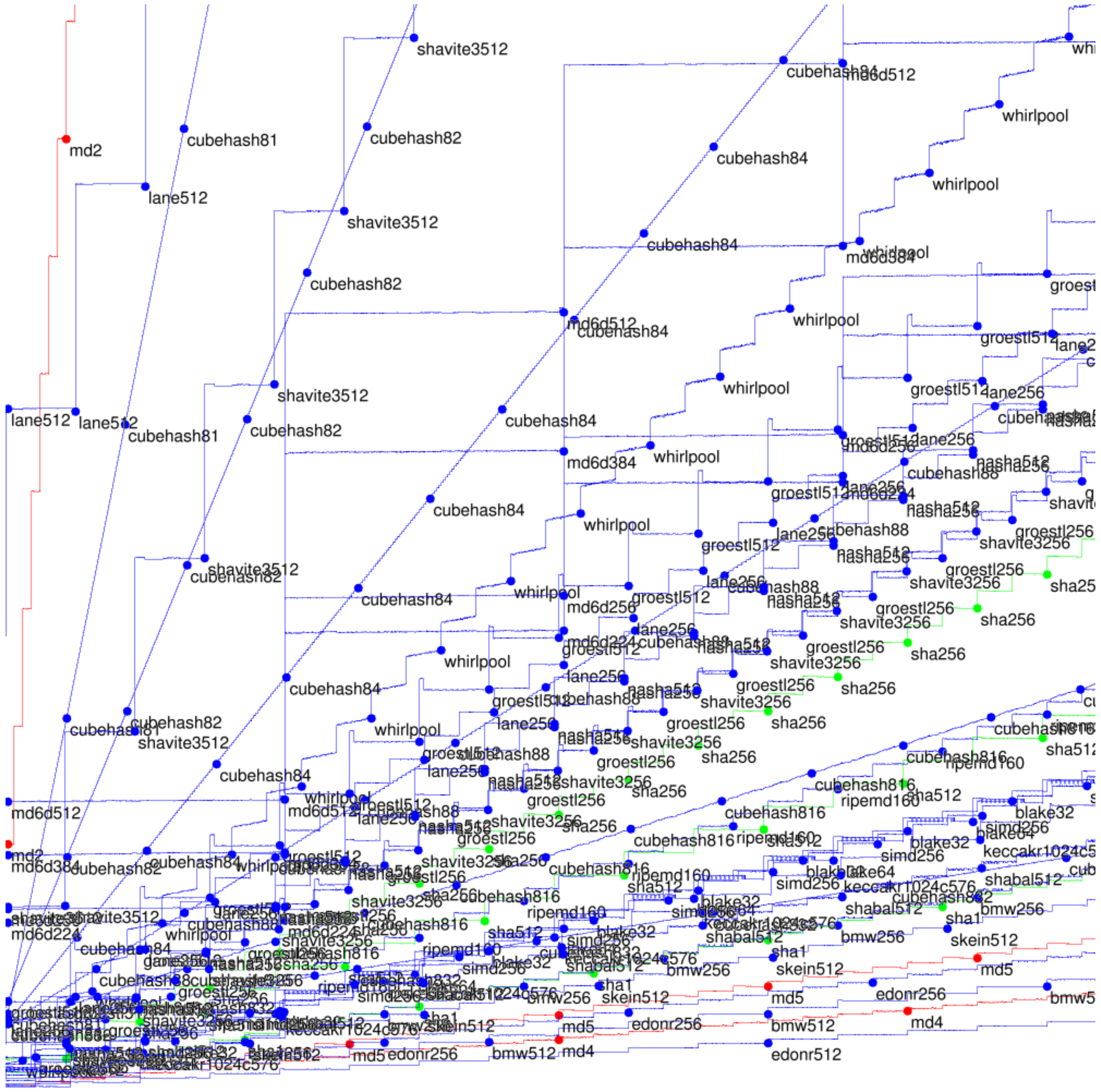
ions.

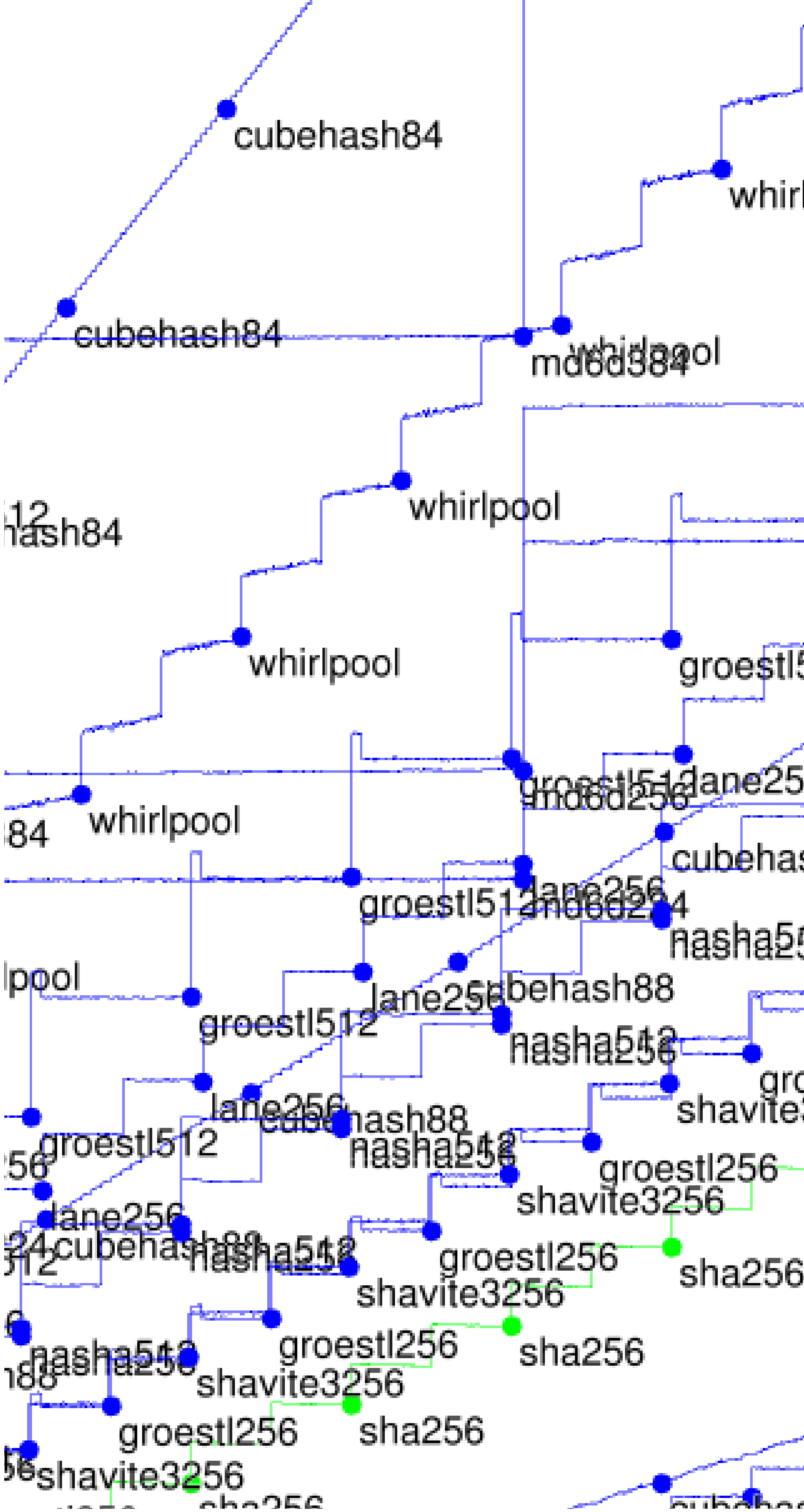
for

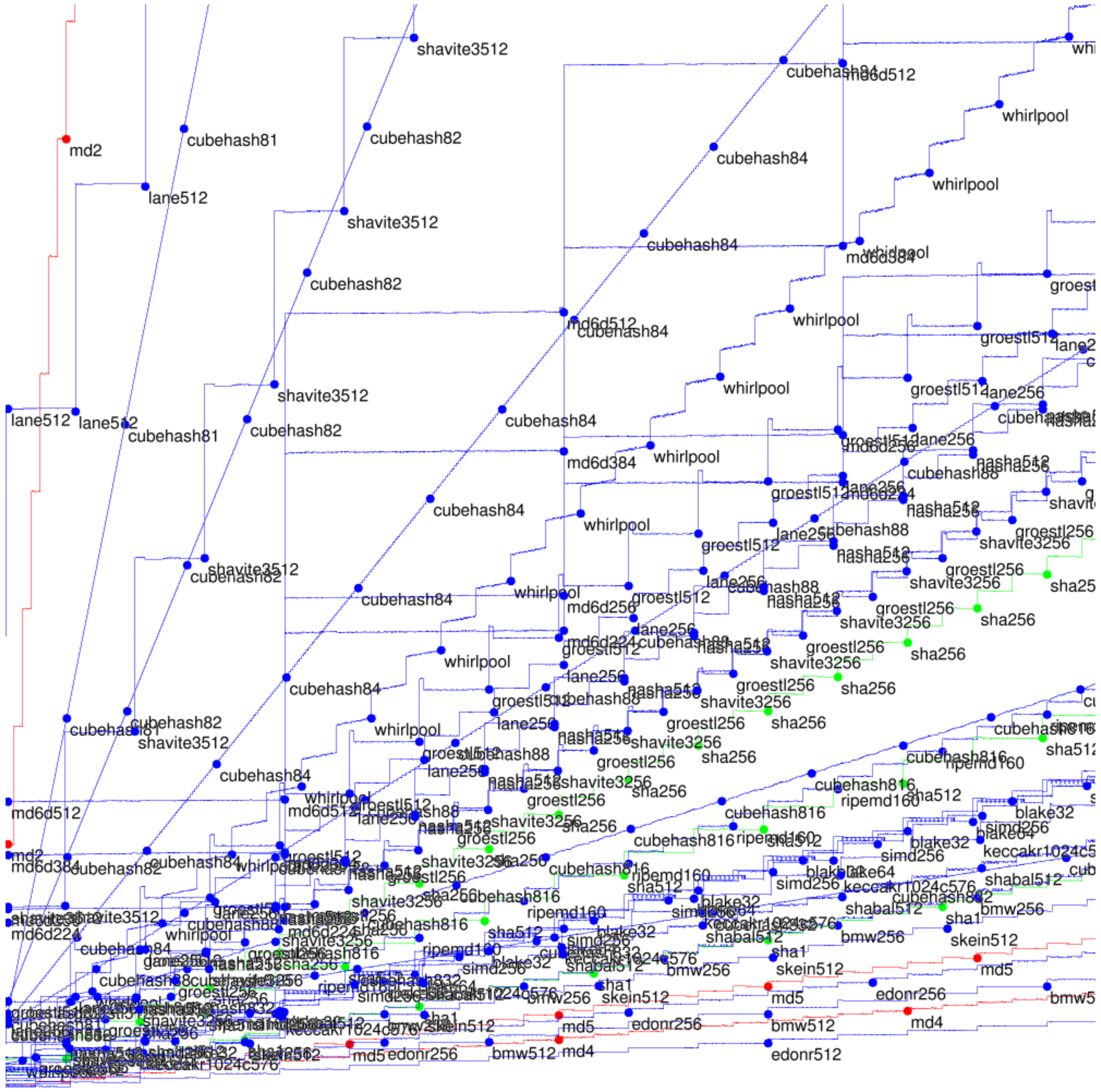
ssage,

ssage,

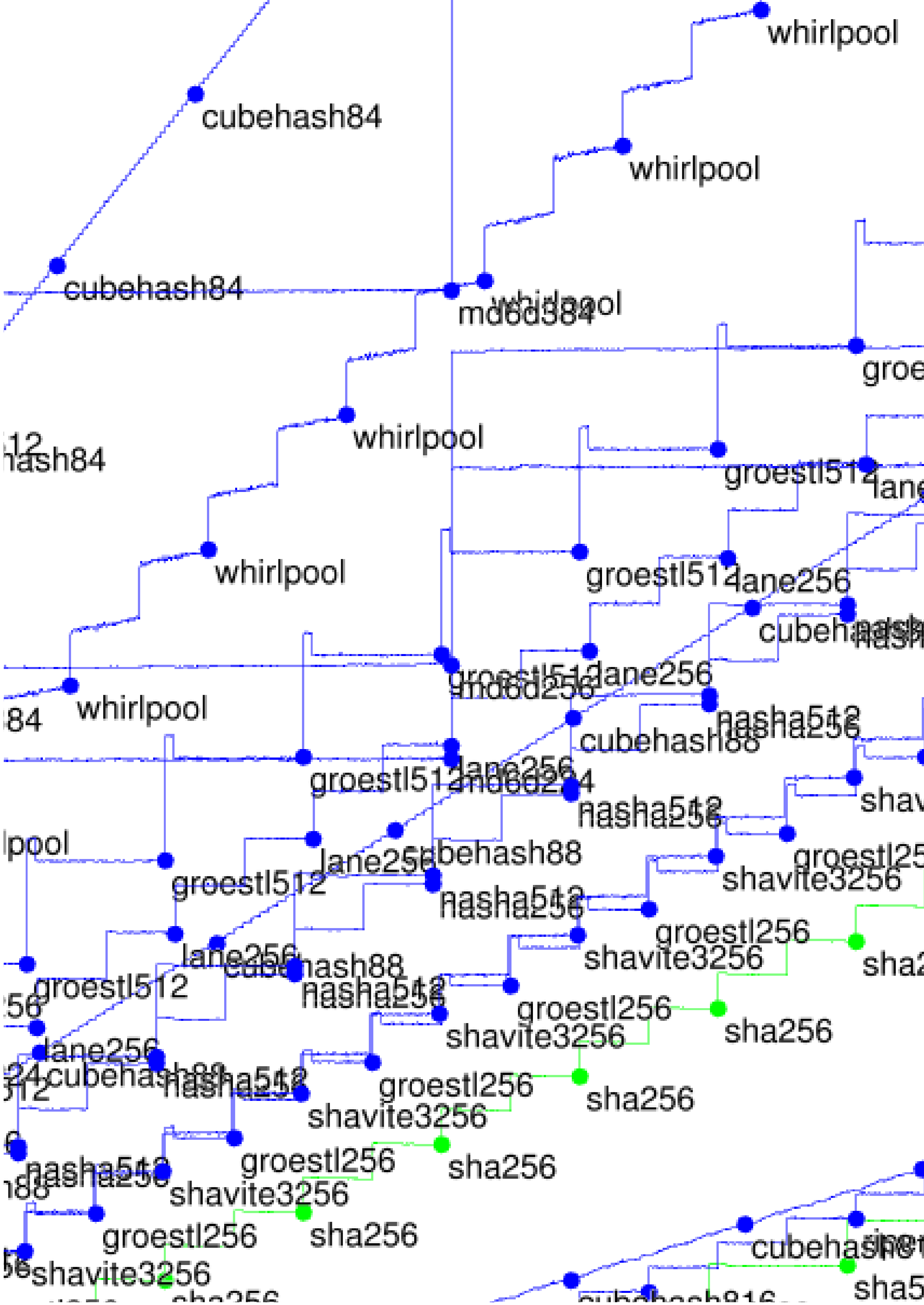
ssage,



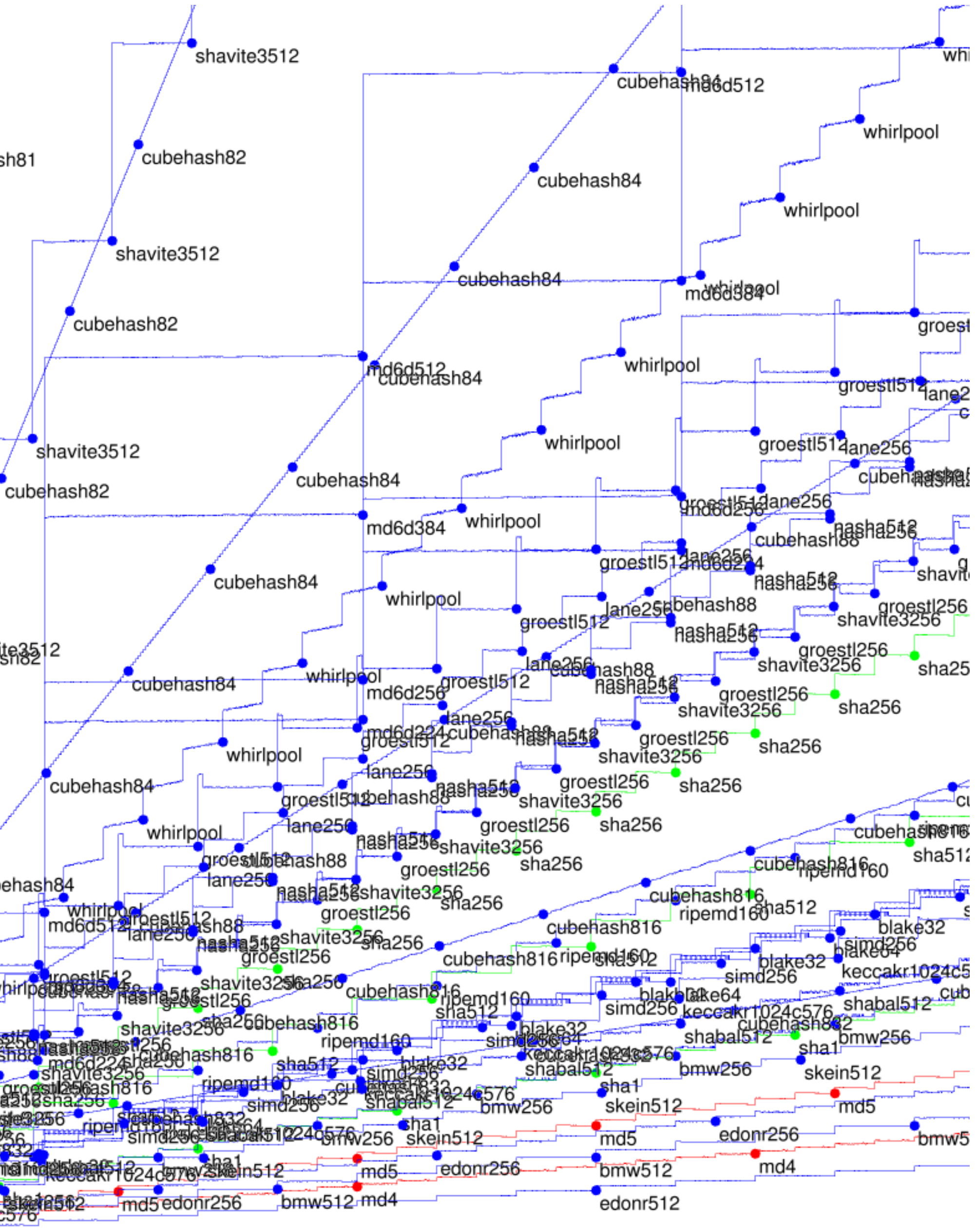


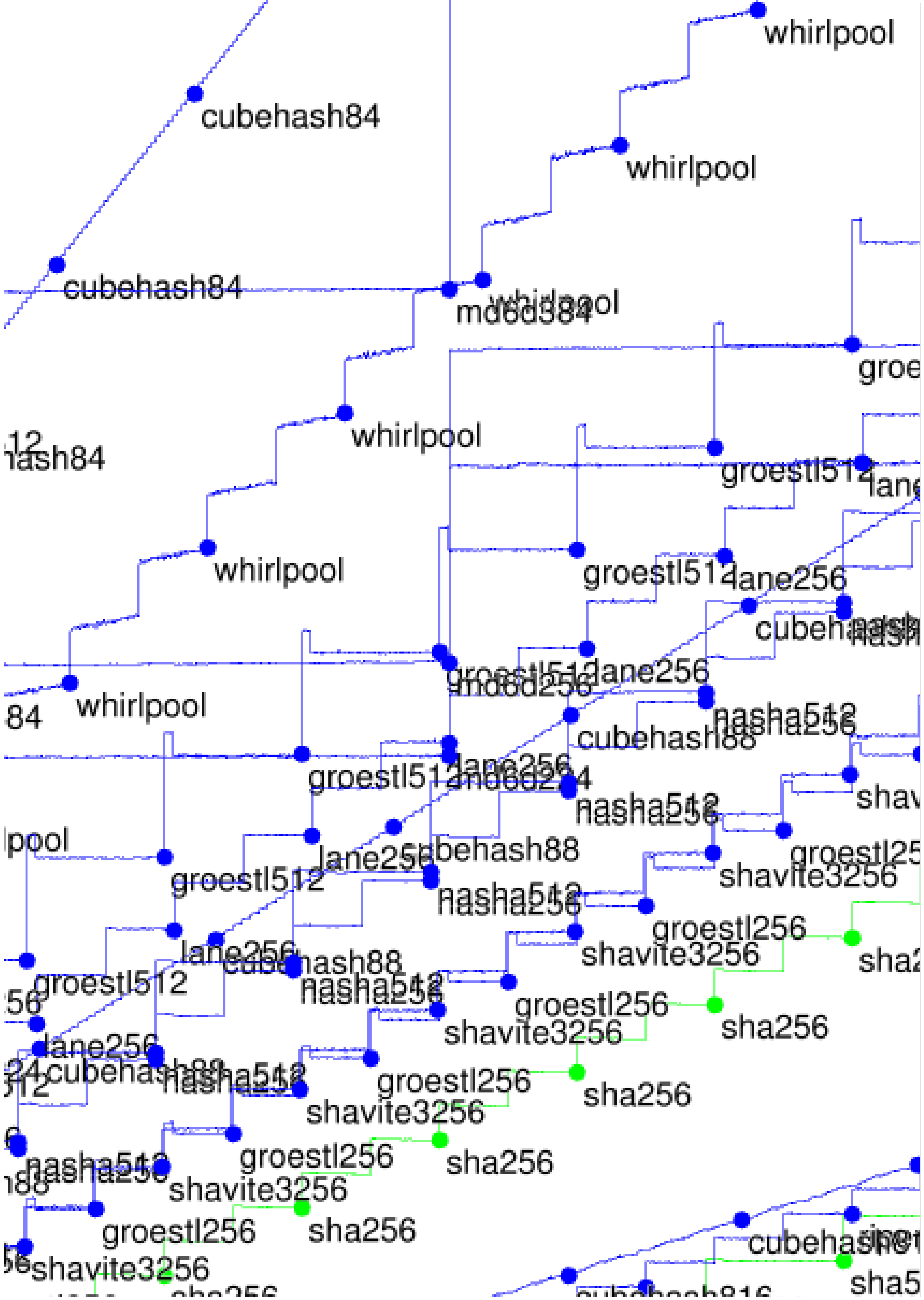


md2
cubehash81
cubehash82
shavite3512
lane512
cubehash84
md6d512
whirlpool
md6d384
groestl512
lane256
cubehash88
nasha512
nasha256
shavite3256
groestl256
sha256
md6d256
md6d224
cubehash816
ripemd160
sha512
blake32
blake64
simd256
simd512
keccakr1024c576
shabal512
bmw256
sha1
skein512
md5
edonr256
bmw512
md4
edonr512



whirlpool
cubehash84
md6d384
groestl512
lane256
md6d256
cubehash88
nasha512
nasha256
shavite3256
groestl256
sha256
cubehash816

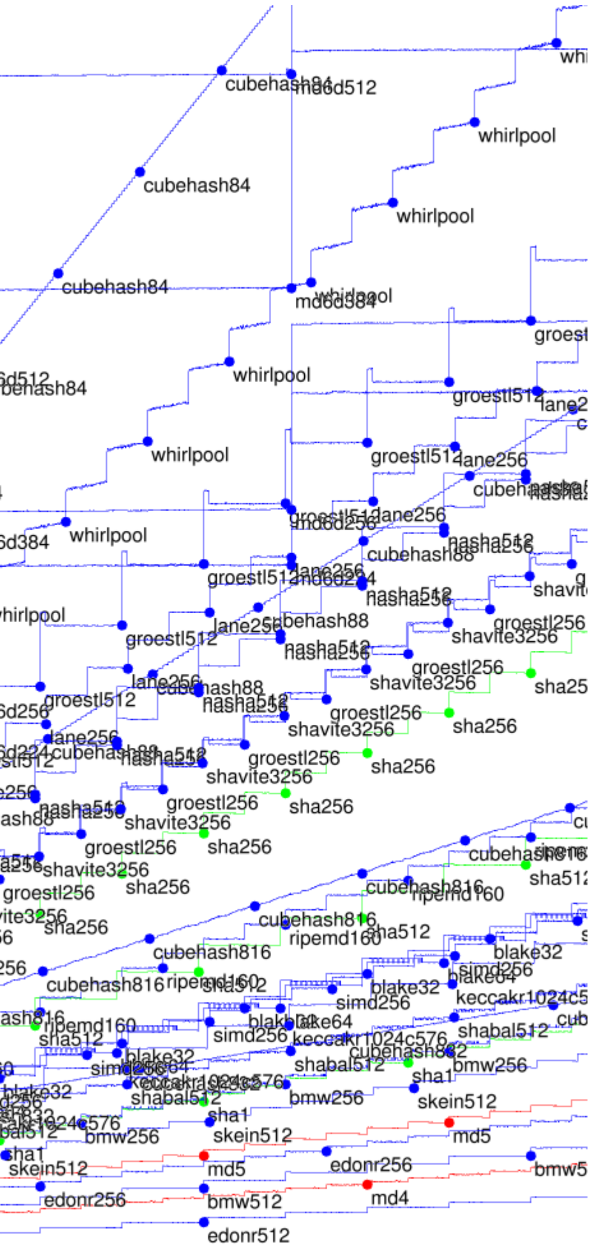


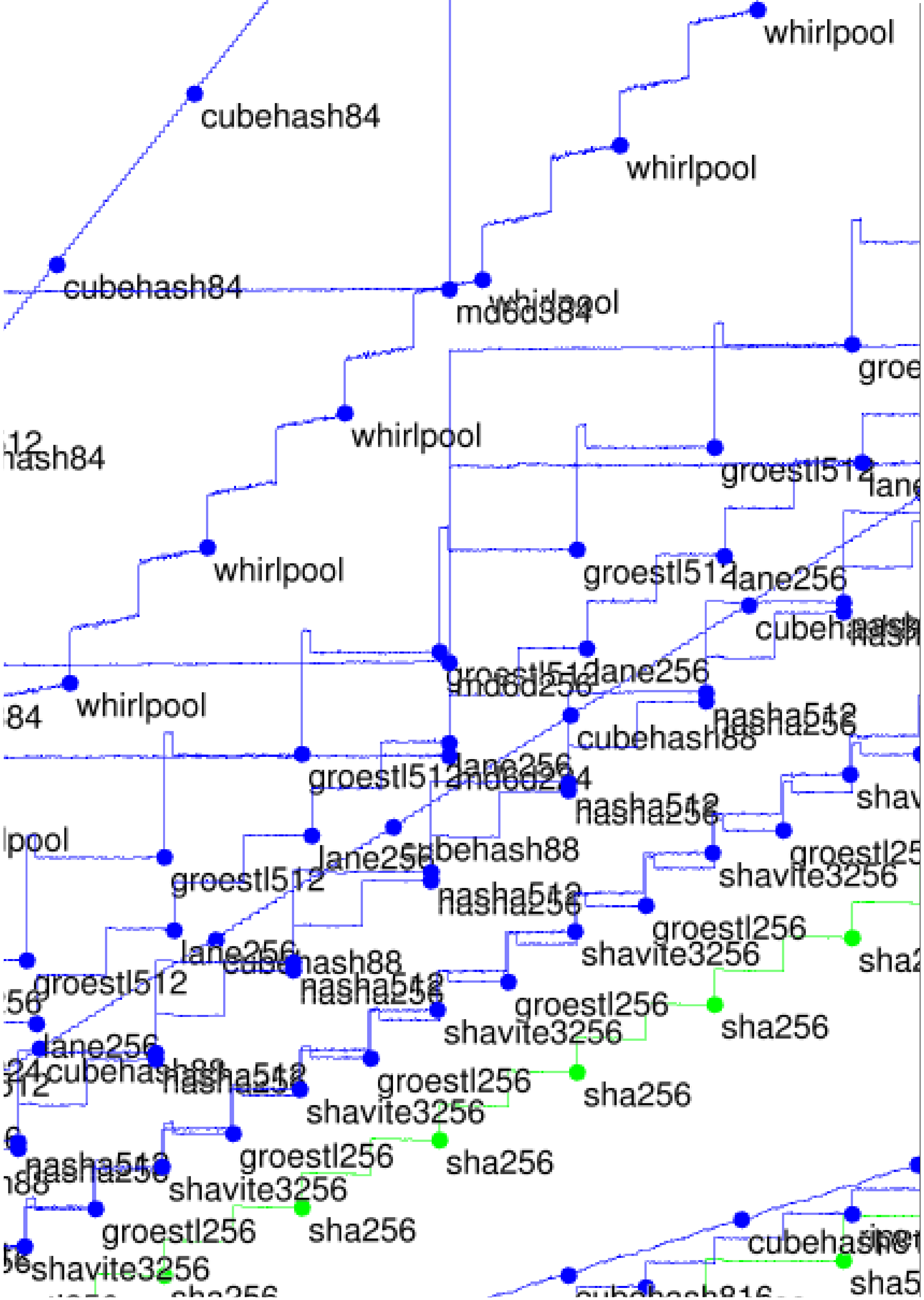


Submit

Define

```
#defi
```

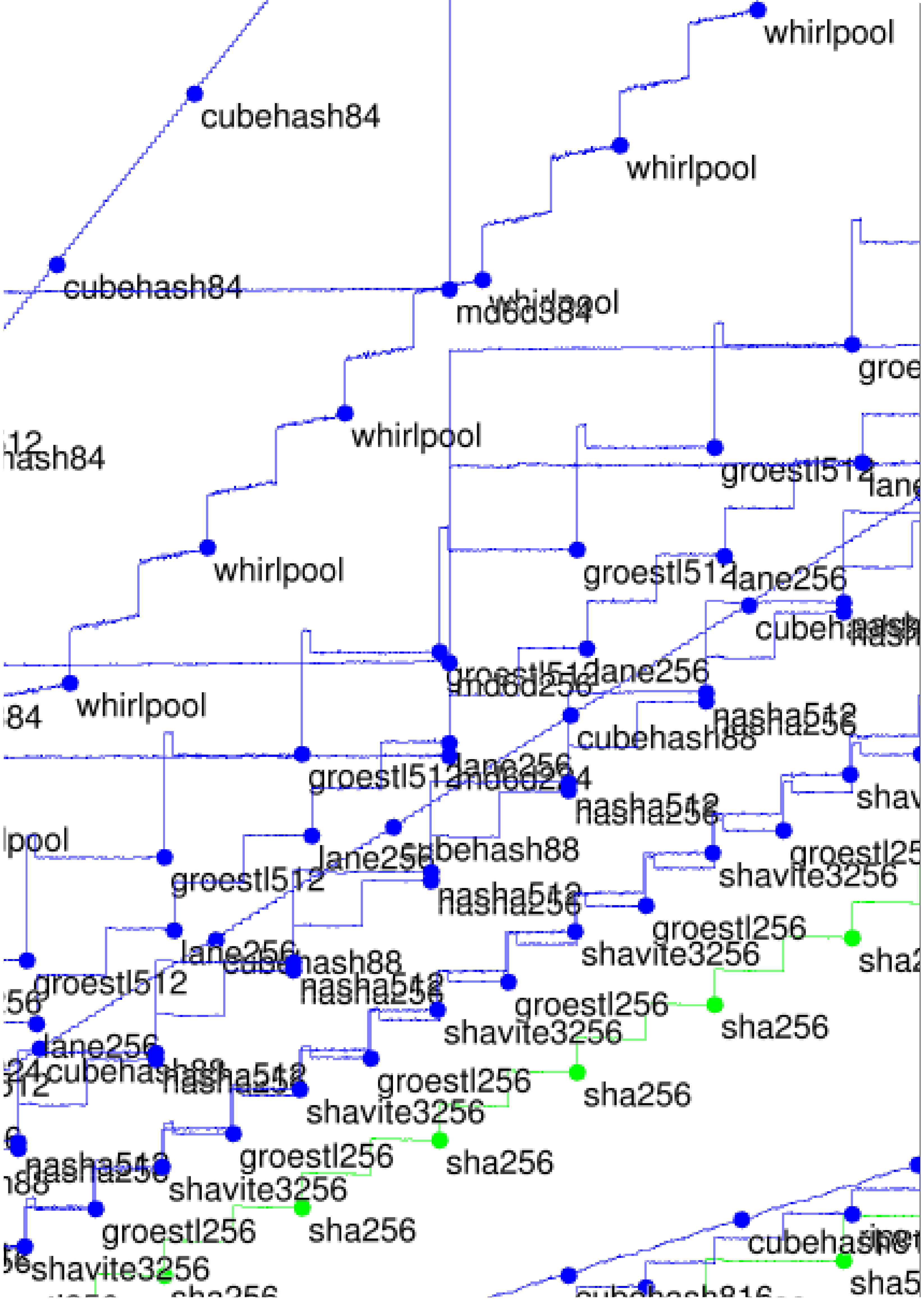




Submitter → eBA

Define output siz

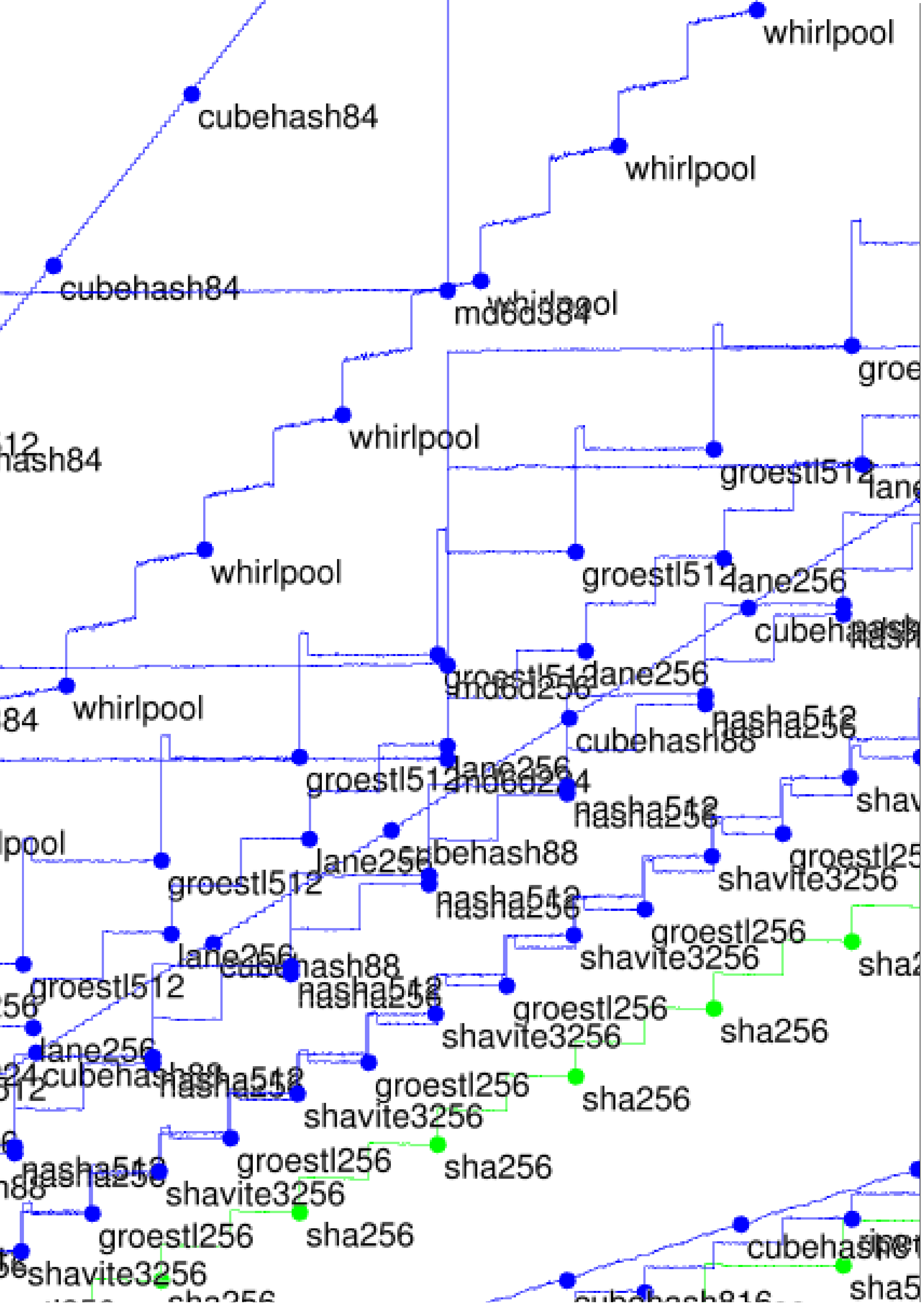
```
#define CRYPT
```



Submitter → eBASH

Define output size in api.

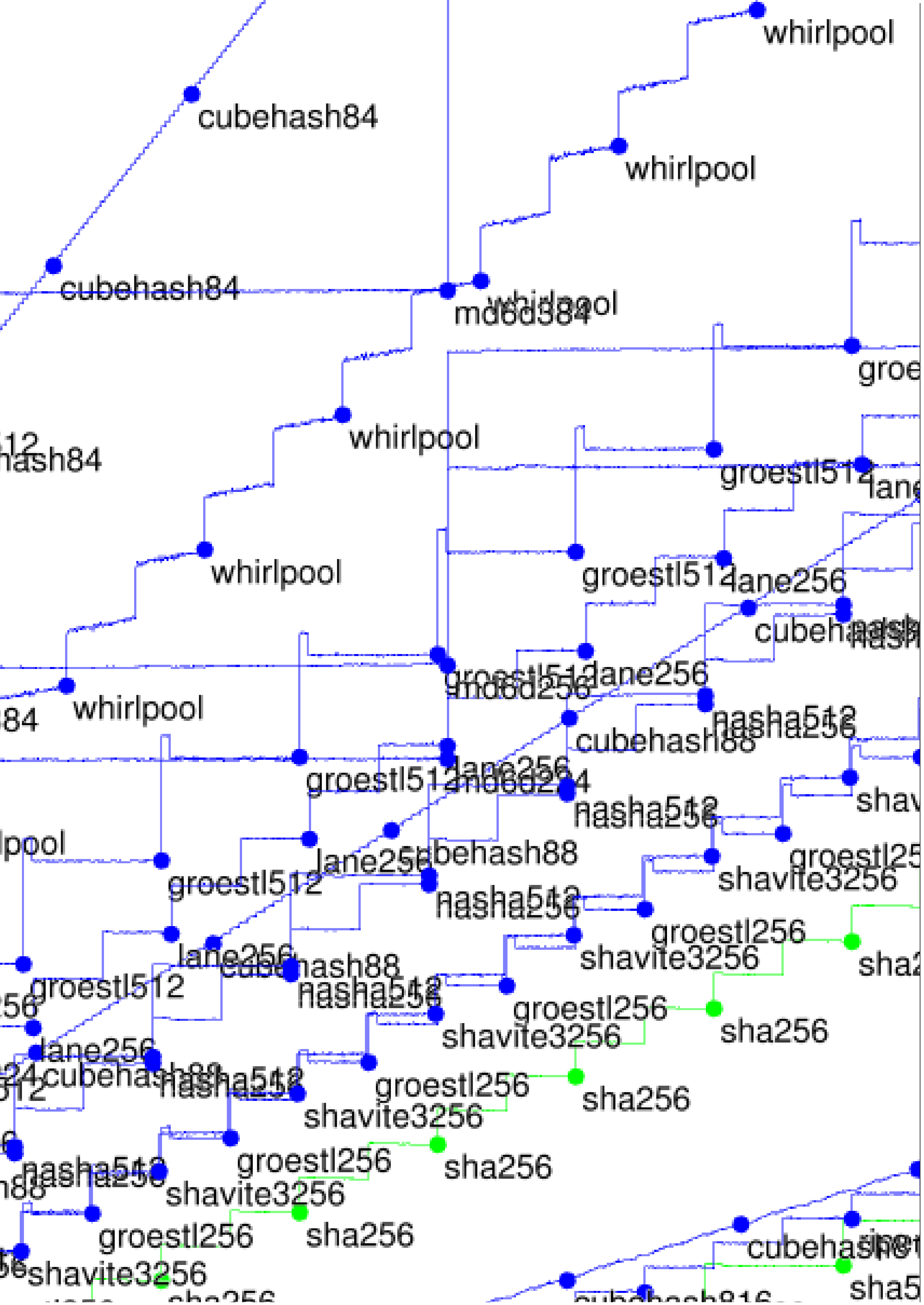
```
#define CRYPTO_BYTES
```



Submitter → eBASH

Define output size in `api.h`:

```
#define CRYPTO_BYTES 64
```



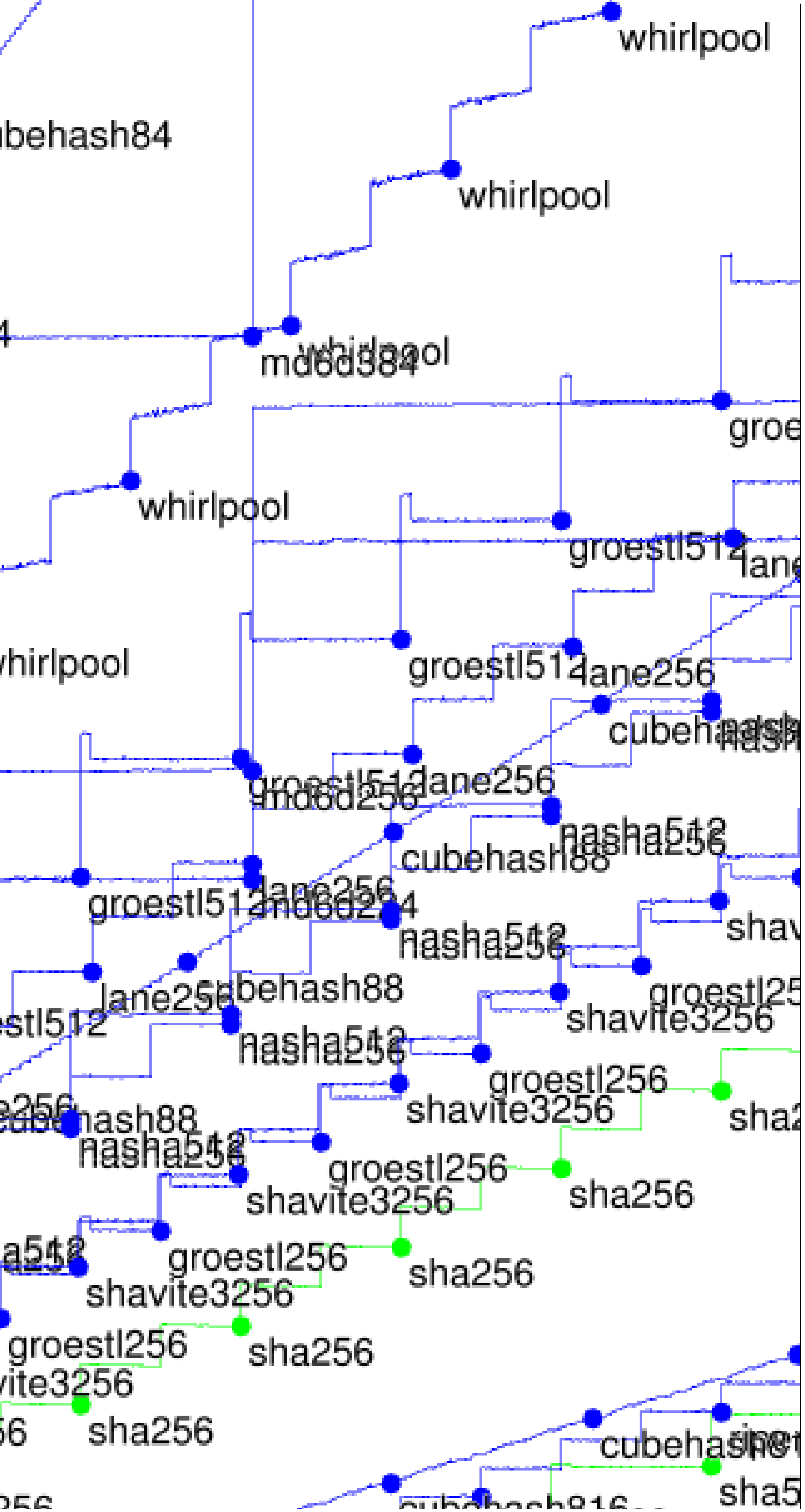
Submitter → eBASH

Define output size in api.h:

```
#define CRYPTO_BYTES 64
```

Define hash function in hash.c, e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
       ,in,inlen*8,out);
  return 0; }
```



Submitter → eBASH

Define output size in api.h:

```
#define CRYPTO_BYTES 64
```

Define hash function in hash.c, e.g. wrapping existing NIST API:

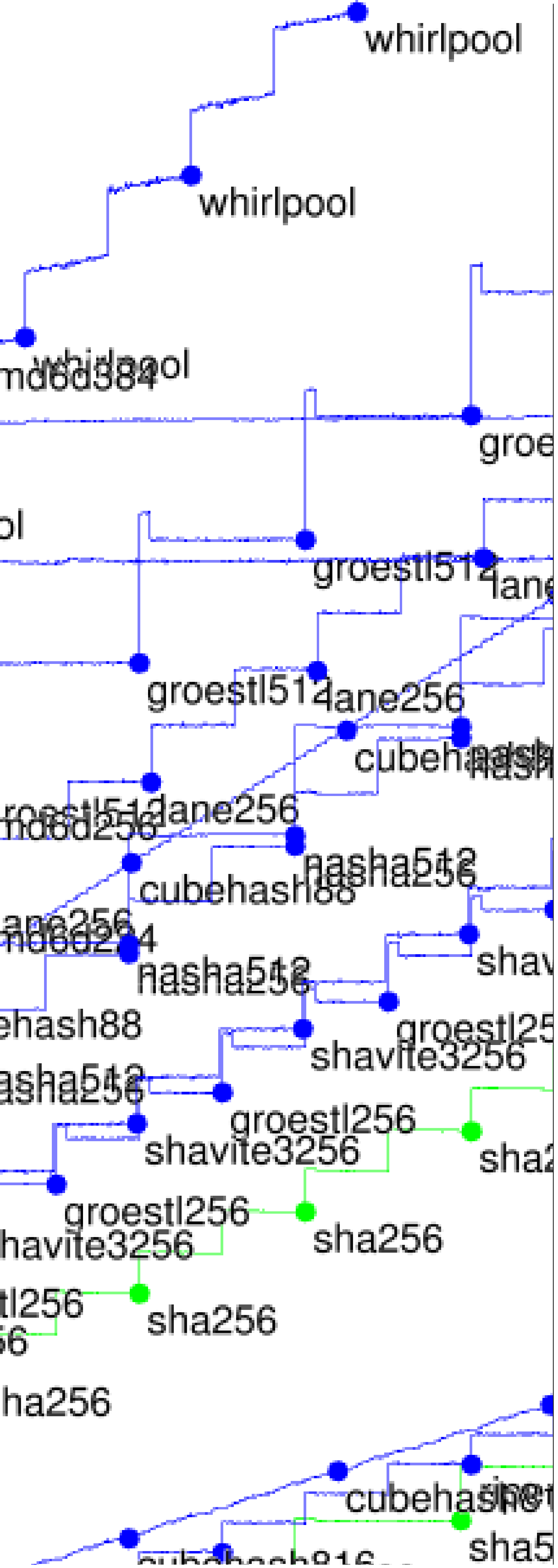
```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
          ,in,inlen*8,out);
  return 0; }
```

Send t
the UR
with o
crypto
contai

Measu
Much e
to do y

More d
http:/
/call-

Also ea
to run



Submitter → eBASH

Define output size in api.h:

```
#define CRYPTO_BYTES 64
```

Define hash function in hash.c, e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
         ,in,inlen*8,out);
  return 0; }
```

Send to the mail
the URL of a ta
with one director
crypto_hash/yo
containing hash.

Measurements m
Much easier than
to do your own b

More details and
http://bench.c
/call-hash.htm

Also easy for thir
to run the bench

Submitter → eBASH

Define output size in `api.h`:

```
#define CRYPTO_BYTES 64
```

Define hash function in `hash.c`, e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
       ,in,inlen*8,out);
  return 0; }
```

Send to the mailing list the URL of a `tar.gz` with one directory `crypto_hash/yourhash/r` containing `hash.c` etc.

Measurements magically ap

Much easier than trying to do your own benchmark

More details and options: http://bench.cr.yp.to /call-hash.html

Also easy for third parties to run the benchmark suite

Submitter → eBASH

Define output size in api.h:

```
#define CRYPTO_BYTES 64
```

Define hash function in hash.c, e.g. wrapping existing NIST API:

```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
  unsigned char *out,
  const unsigned char *in,
  unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
       ,in,inlen*8,out);
  return 0; }
```

Send to the mailing list the URL of a tar.gz with one directory crypto_hash/yourhash/ref containing hash.c etc.

Measurements magically appear! Much easier than trying to do your own benchmarks.

More details and options: http://bench.cr.yp.to/call-hash.html

Also easy for third parties to run the benchmark suite.