

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

<http://bench.cr.yp.to>

n.html

Bernstein

University of Illinois at Chicago

work with:

– ange

Netherlands 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.

ECRYPT

STVL,

include

STVL

ran eS

De Car

eSTRE

Stream

matchi

were co

publish

benchm

e.g. 18

third-p

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, symmetric included four wo STVL WG 1, stre ran eSTREAM (2

De Cannière *pub* eSTREAM bench Stream-cipher im matching the ber were contributed *published*, often benchmarks on e.g. 18 cycles/by third-party asm A

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” STVL, symmetric-technique included four working groups STVL WG 1, stream-cipher ran eSTREAM (2004–2008)

De Cannière *published* eSTREAM benchmarking tool

Stream-cipher implementations matching the benchmarking tool were contributed by design teams *published*, often tuned;

benchmarked on many computers

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

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
3-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: ‘
Applicati
Lab,’ s
(“ECR
of Asyn
measur
encrypt

Publish

Have w
49 pub
matchi
Benchm

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

nnance evaluators

nterations

graphic systems,

the same API;

arketing toolkit;

n 25 computers.

erformance results:

byte 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 Bench
of Asymmetric S
measuring efficie
encryption, signa

Published a new

Have written, co

49 public-key im

matching the ben

Benchmarked on

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 encryption, signatures, DH

Published a new toolkit.

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

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,
and four working groups.

WG 1, stream-cipher group,
TREAM (2004–2008).

annière *published*
CAM benchmarking toolkit.

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: ‘
 (“ECR”
of Stre
post-eS
VAMP
 (“ECR”
of All S
eBACS
of Cryp
include
Contin
New to
with C
AES no

ive “virtual labs.”
c-techniques lab,
working groups.

team-cipher group,
2004–2008).

Published
benchmarking toolkit.

implementations
benchmarking 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 Benchmarks of Stream Ciphers post-eSTREAM”)
VAMPIRE also started
 (“ECRYPT Benchmarking of All Submitted Cryptographic Algorithms”)
eBACS (“ECRYPT Benchmarking of Cryptographic Algorithms”) includes eBATS, CACE, and others.
Continues under the name eBACS.
New toolkit, API, and benchmarks with CACE library.
AES now 14 cycles.

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 eSTREAM (“ECRYPT Benchmarking of Stream Ciphers”) for post-eSTREAM benchmarking

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

eBACS (“ECRYPT Benchmark of Cryptographic Systems” includes eBATS, eBASH, eBCH, eBES, eBESI, eBESII. Continues under ECRYPT II.

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

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
CRYPT Benchmarking
ymmetric Systems”),
ring efficiency of public-key
tion, signatures, DH.

ned a new toolkit.

written, 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 has

<http://.../result>

already

measur

101 ma

Each i

recomp

with va

to iden

for imp

, “Virtual Implementation ATS
Benchmarking systems”),
ency of public-key
tures, DH.

toolkit.
lected, 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 already
77 implementations
38 hash functions
<http://bench.eccr.be/>
[/results-hash](http://bench.eccr.be/results-hash)
already shows measurements on
101 machine-AB
Each implementation recompiled 1226
with various compilers
to identify best way
for implementation

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 options for implementation, machine,

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](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").

("ECRYPT Benchmarking
ryptographic Systems")
es eBATS, eBASH, eBASC.
ues under ECRYPT II.

oolkit, 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](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	

I started eBASC
Benchmarking
“rs”) for
benchmarks.

I started eBASH
Benchmarking
“Hashes”).

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

; coordinated
try (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](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.

	25%	50%	75%
Duo 6f6, 2137MHz	2.83	2.83	2.83
4.46	4.46	4.46	4.46
5.29	5.30	5.30	5.30
7.08	7.08	7.08	7.08
8.29	8.30	8.30	8.30
8.39	8.39	8.39	8.39
9.59	9.59	9.59	9.59
9.67	9.76	9.76	9.76
11.29	11.30	11.30	11.30
11.47	11.49	11.49	11.49
12.08	12.08	12.08	12.08
12.05	12.09	12.09	12.09
14.83	14.83	14.83	14.83

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 (Core Duo 6f6, 2137MHz), 64-bit

	25%	50%	75%	hash
edonr512	2.83	2.83	2.83	edonr512
bmw512	4.46	4.46	4.46	bmw512
edonr256	5.29	5.30	5.38	edonr256
skein512	7.08	7.08	7.08	skein512
sha1	8.29	8.30	8.30	sha1
bmw256	8.39	8.39	8.47	bmw256
cubehash5	9.59	9.59	9.60	cubehash5
shabal512	9.67	9.76	9.76	shabal512
keccak512	11.29	11.30	11.30	keccak512
simd256	11.47	11.49	11.54	simd256
blake64	12.08	12.08	12.08	blake64
blake32	12.05	12.09	12.09	blake32
sha512	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](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.

I → public

I has already collected implementations of hash functions in 18 families.

//bench.cr.yp.to

lts-hash.html

shows

measurements on 71 machines; machine-ABI combinations.

Implementation is

compiled 1226 times

various compiler options

to identify best working option

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.

Tables

of cycle

8-byte

64-byte

576-by

1536-b

4096-b

(extrap

Actuall

e.g. Re

e.g. Gr

0-byte

2-byte

4-byte

..., 20

dy collected
ns 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 message
1536-byte message
4096-byte message
(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

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, etc.
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
0-byte message, 1-byte message
2-byte message, 3-byte message
4-byte message, 5-byte message
..., 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.

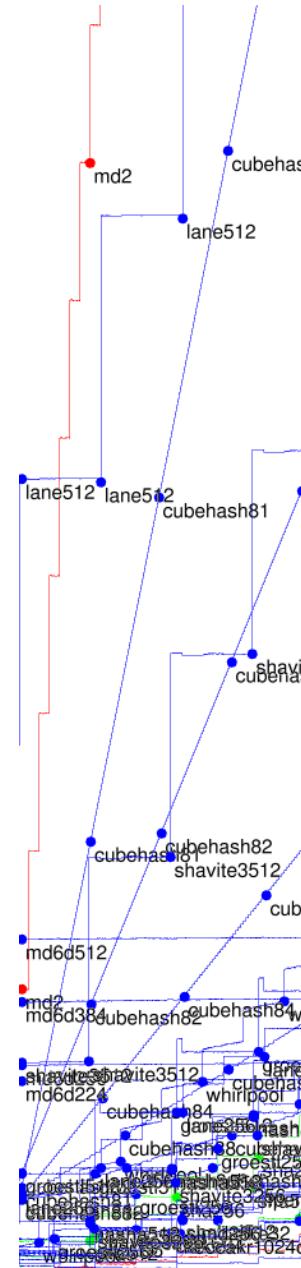
e.g. Graphs show medians for
0-byte message, 1-byte message,
2-byte message, 3-byte message,
4-byte message, 5-byte message,
. . . , 2048-byte message.

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.

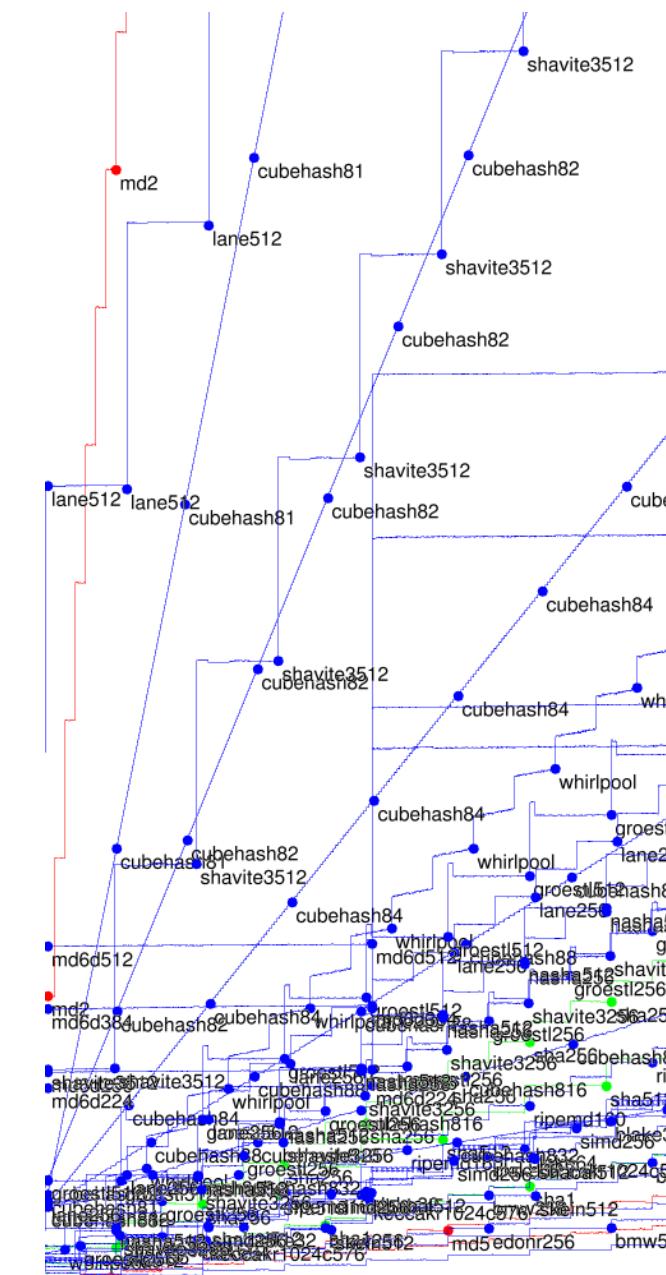


katana (Core 2
Hz), 64-bit ABI:

%	hash
33	edonr512
46	bmw512
38	edonr256
08	skein512
30	sha1
47	bmw256
50	cubehash832
76	shabal512
30	keccak1024c576
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.

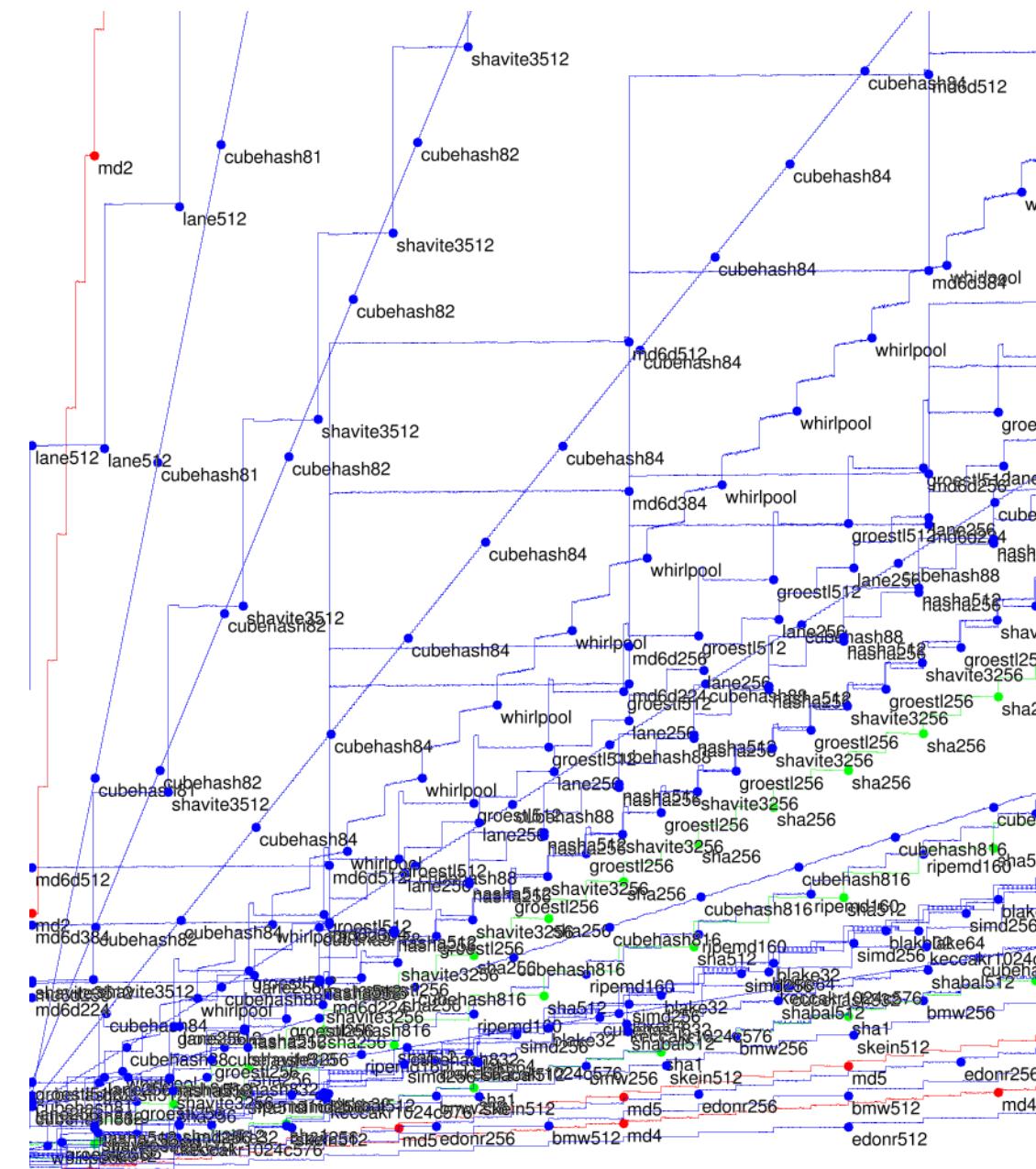
Actually have much more data.
e.g. Reports show best options.
e.g. Graphs show medians for
0-byte message, 1-byte message,
2-byte message, 3-byte message,
4-byte message, 5-byte message,
. . . , 2048-byte message.



12
2
56
2
56
sh832
512
r1024c576
6
4
2

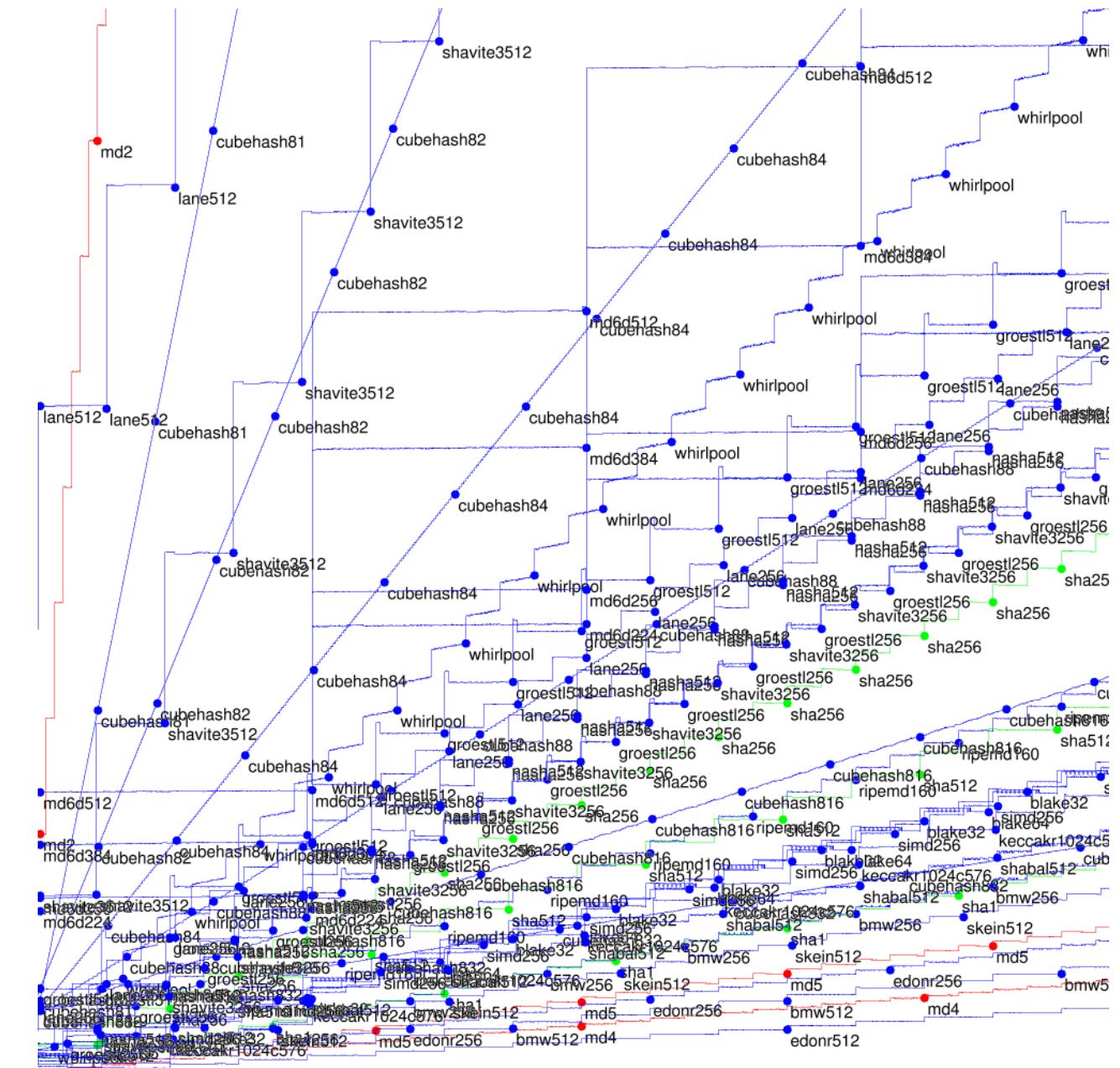
Tables show medians, quartiles
of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long message.

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.



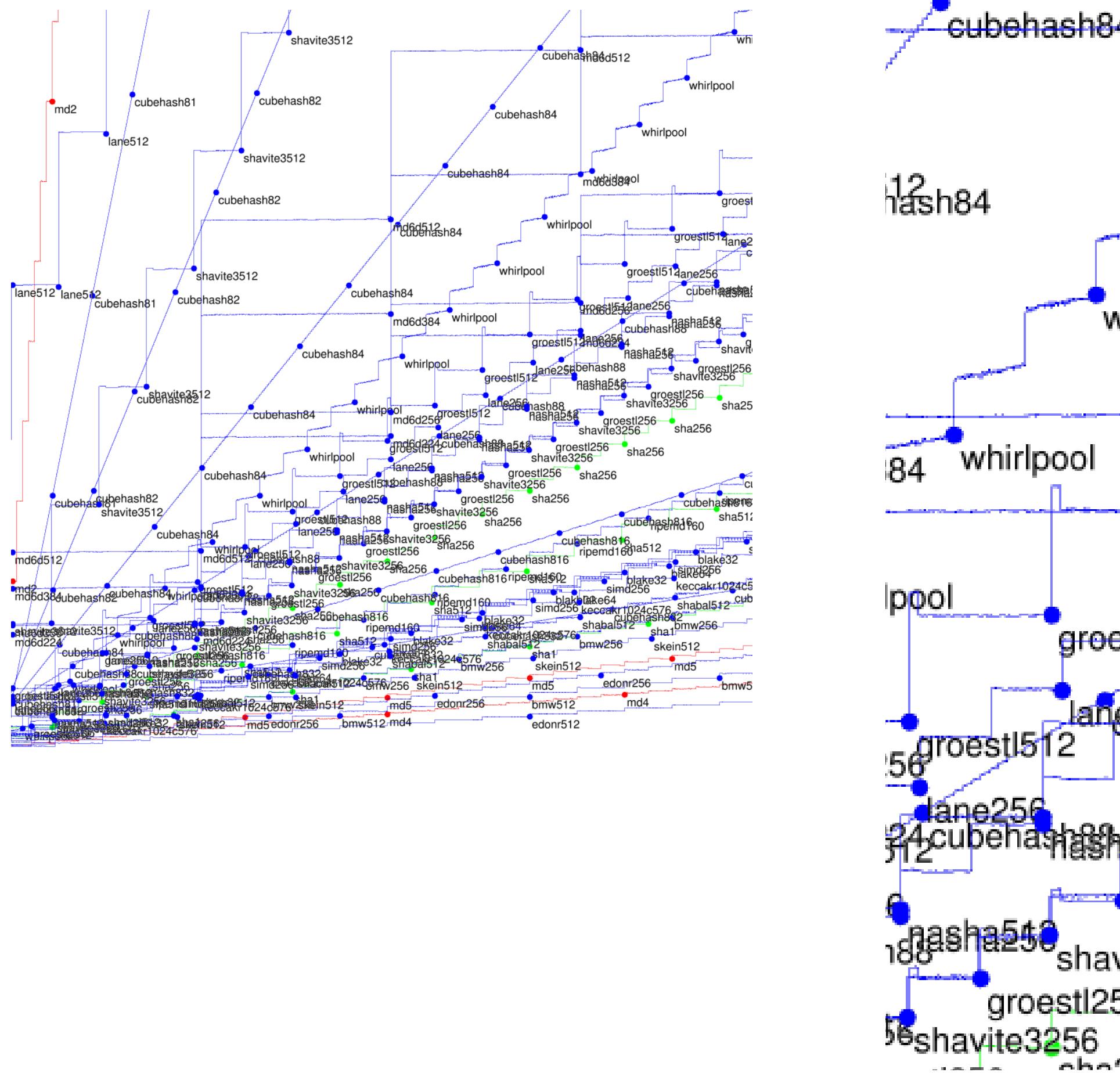
Tables show medians, quartiles
of cycles/byte to hash
8-byte message,
64-byte message,
576-byte message,
1536-byte message,
4096-byte message,
(extrapolated) long message.

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.
reports show best options.
aphs show medians for
message, 1-byte message,
message, 3-byte message,
message, 5-byte message,
48-byte message.



ilians, quartiles

hash

e,
ge,
ge,
ng message.

uch more data.

new best options.

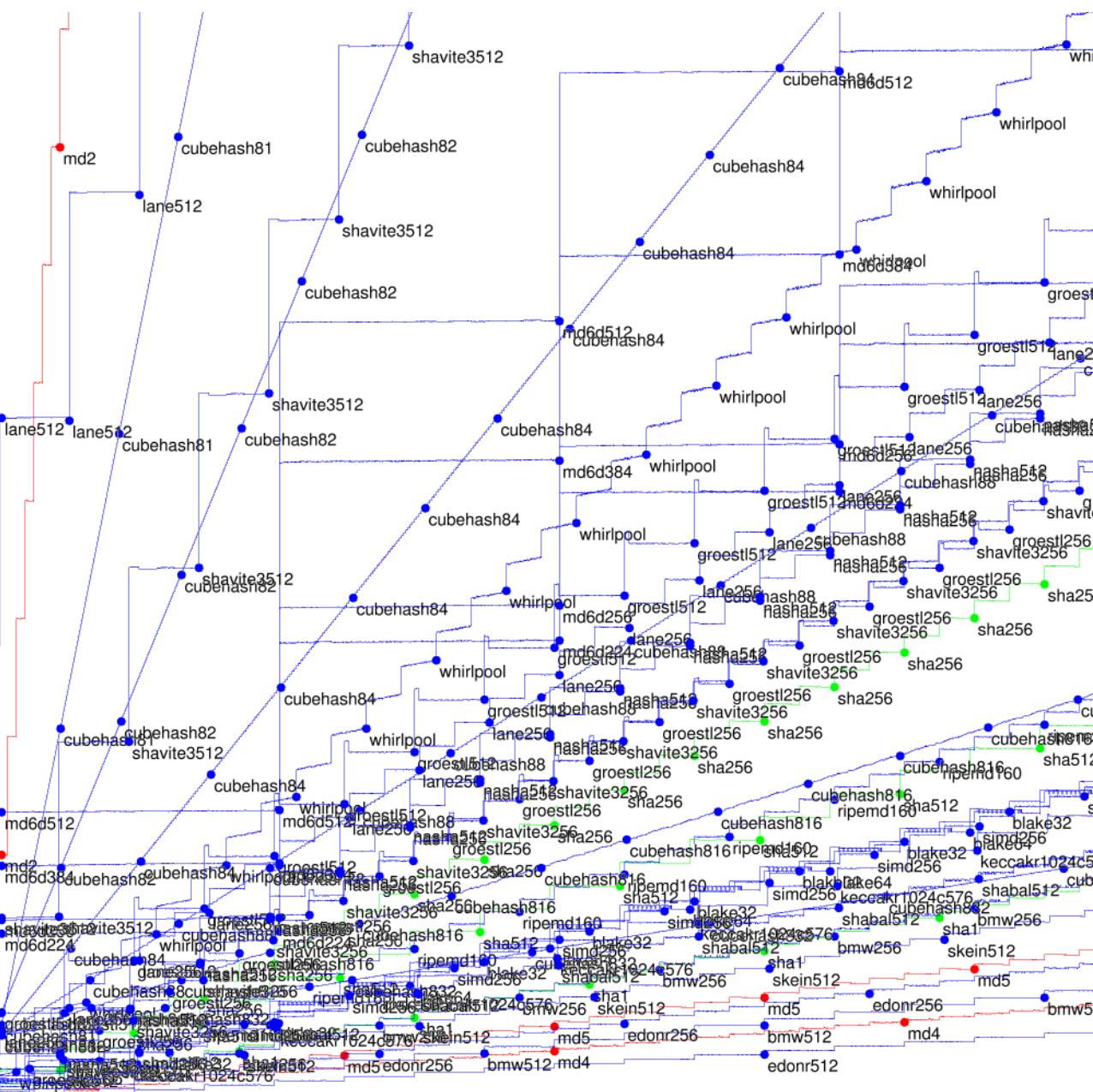
γ medians for

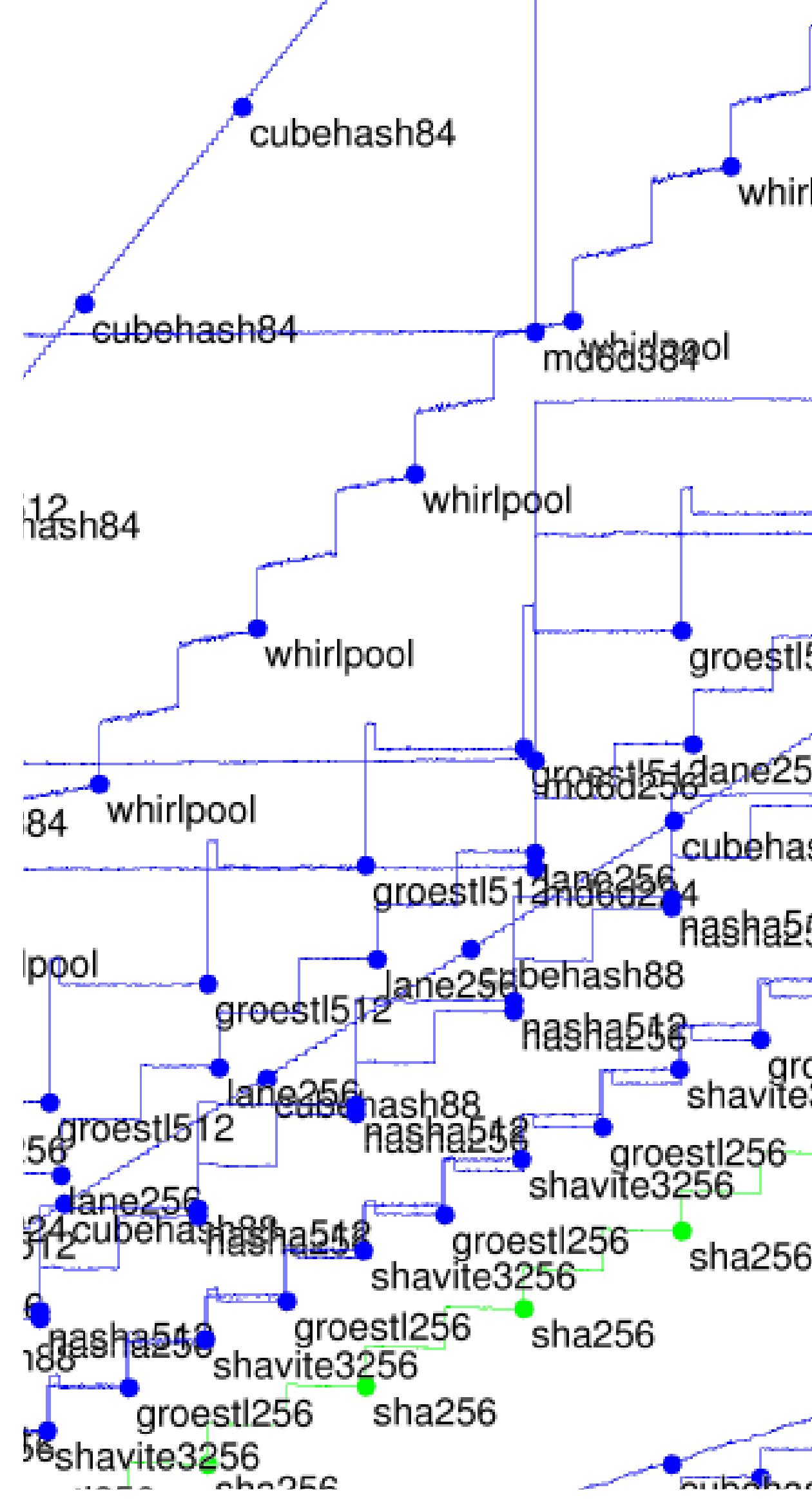
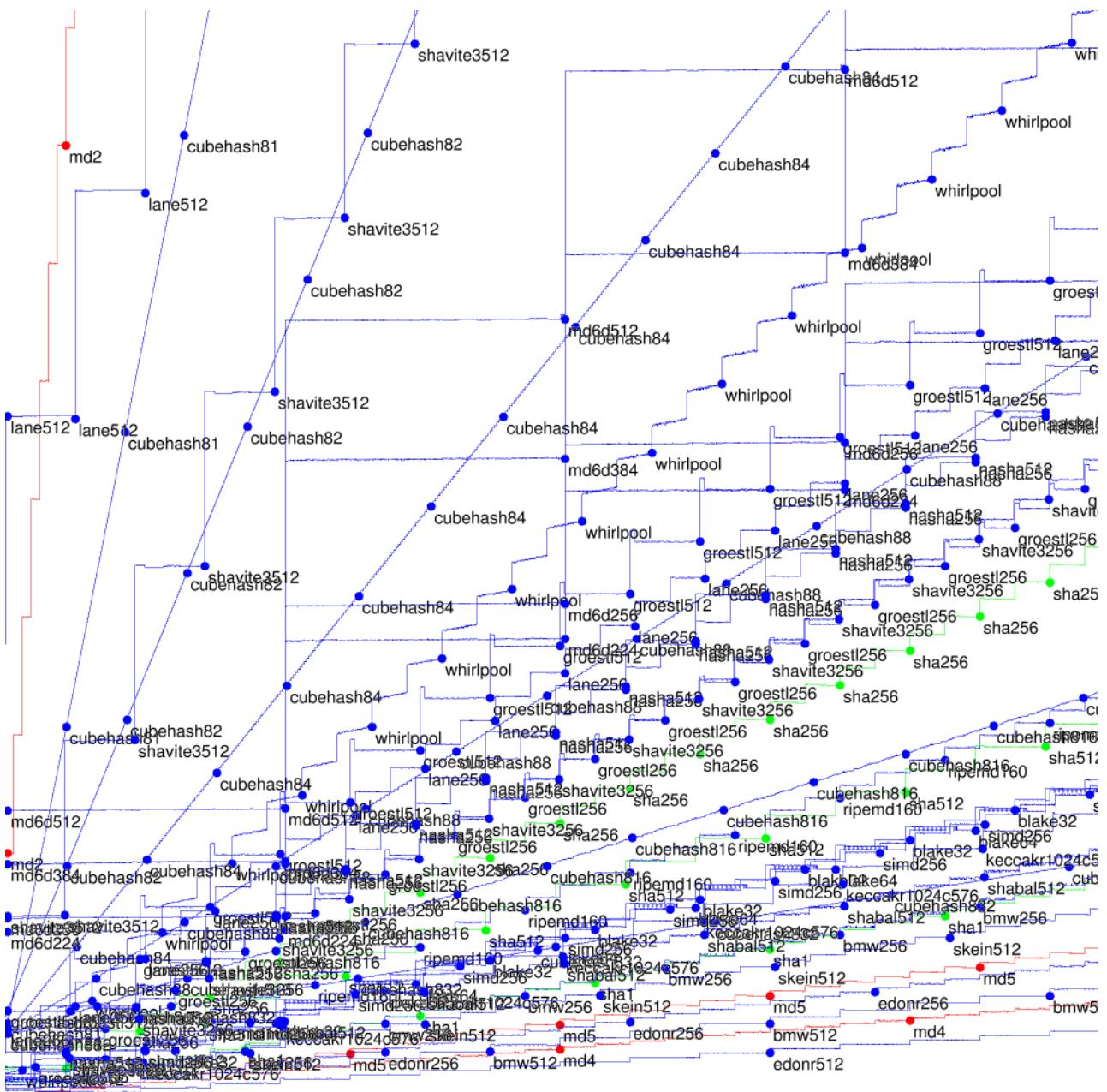
1-byte message,

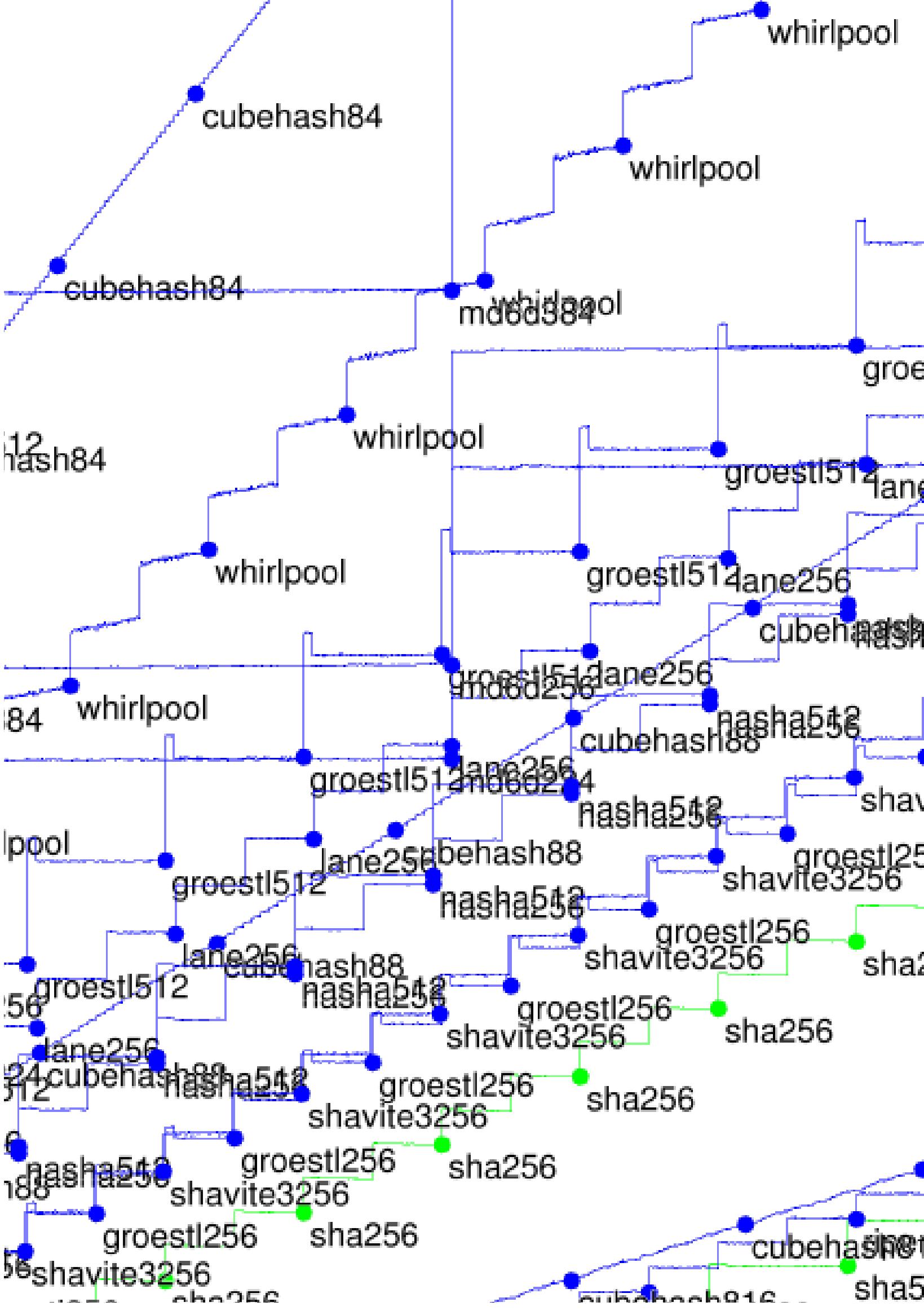
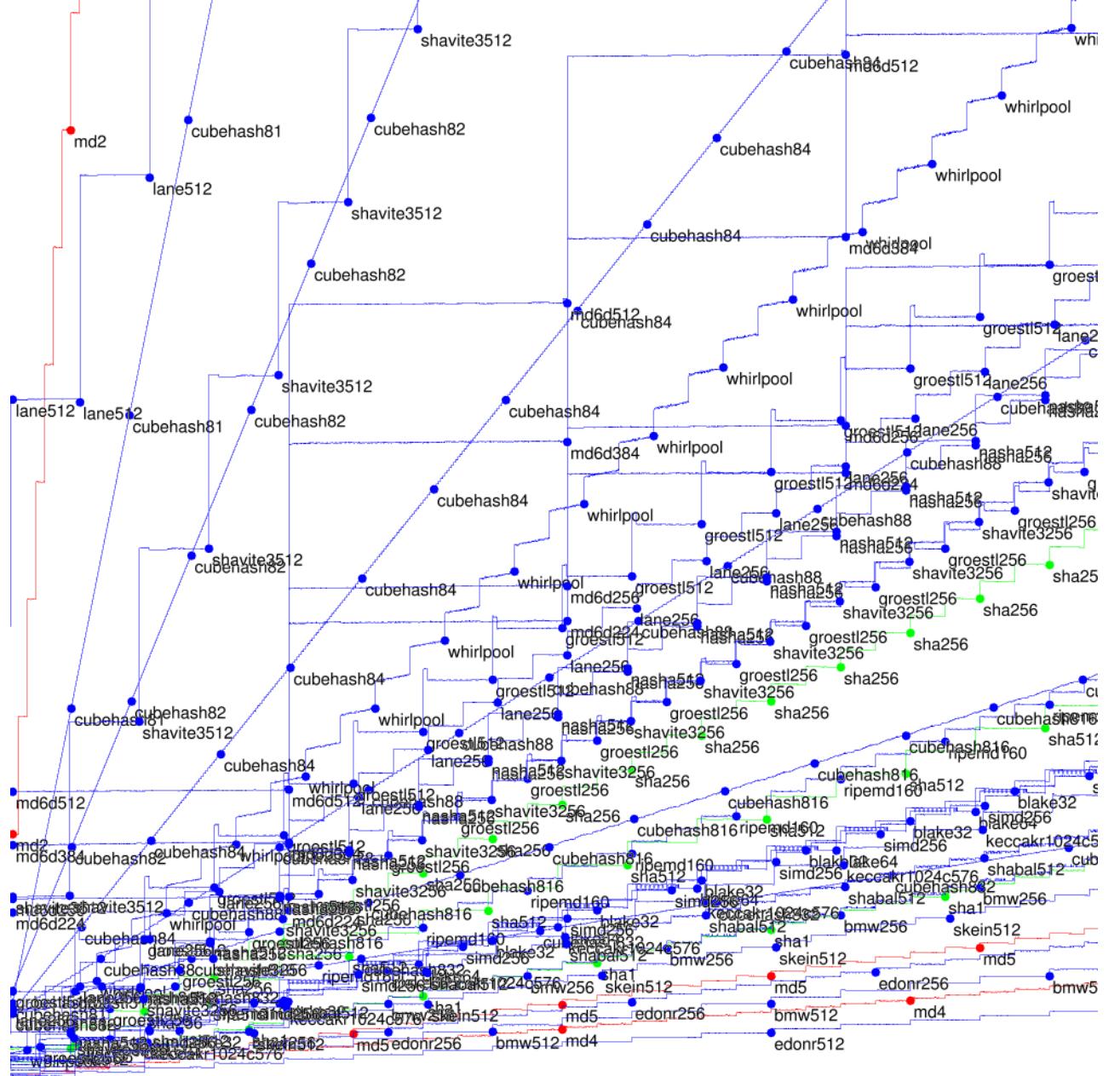
3-byte message,

5-byte message,

essage.



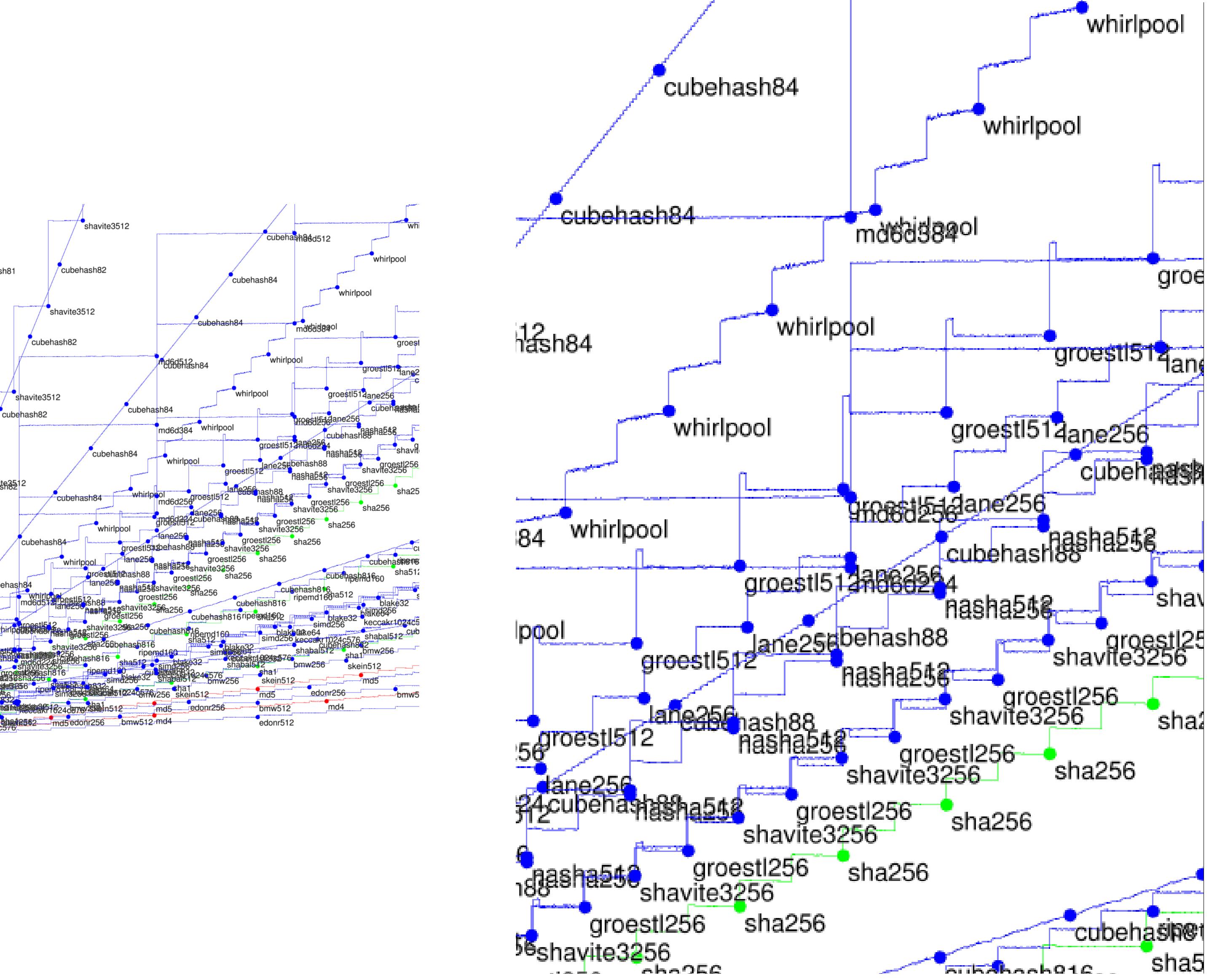




Submit

Define

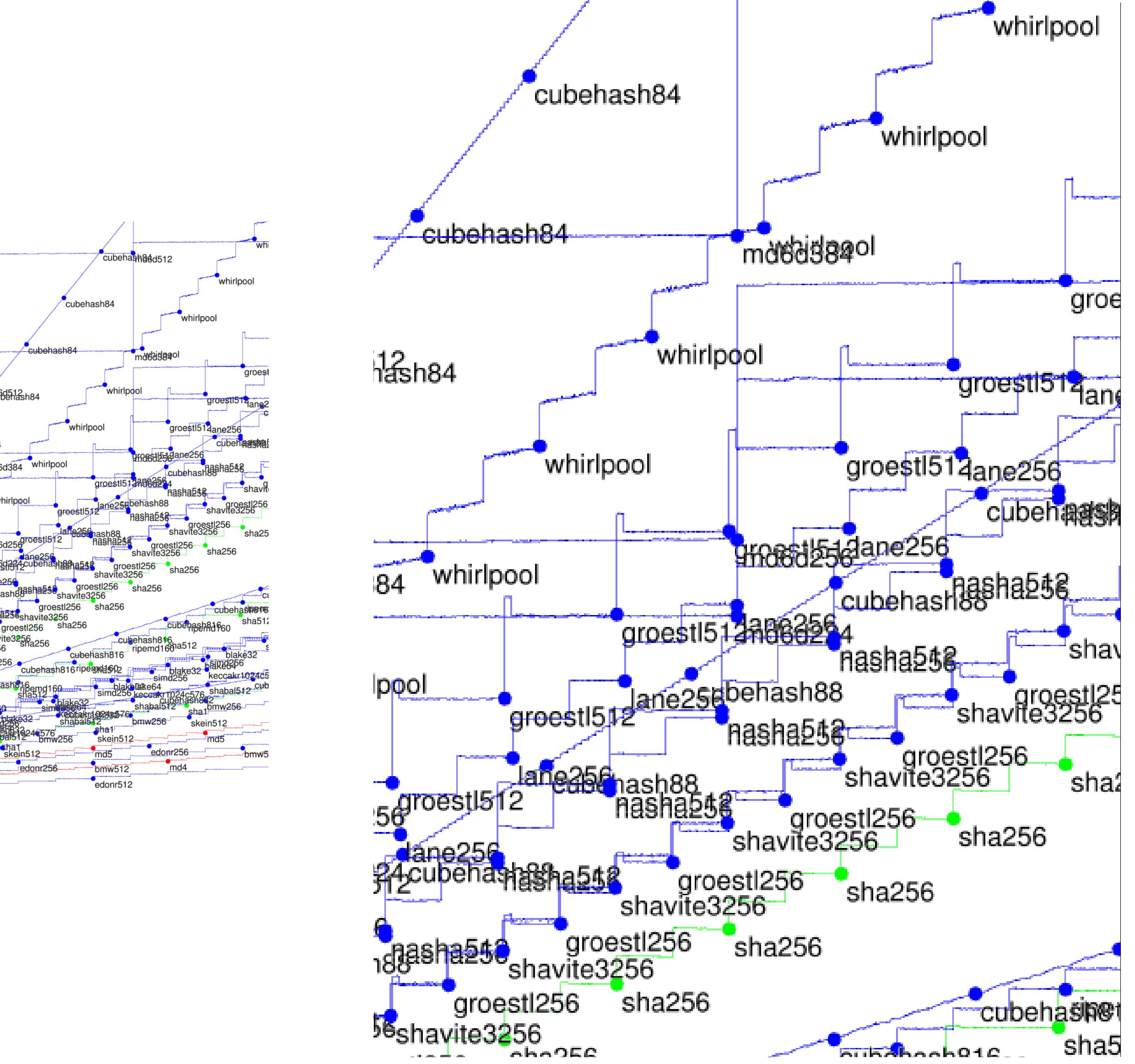
#def



Submitter → eBA

Define output size

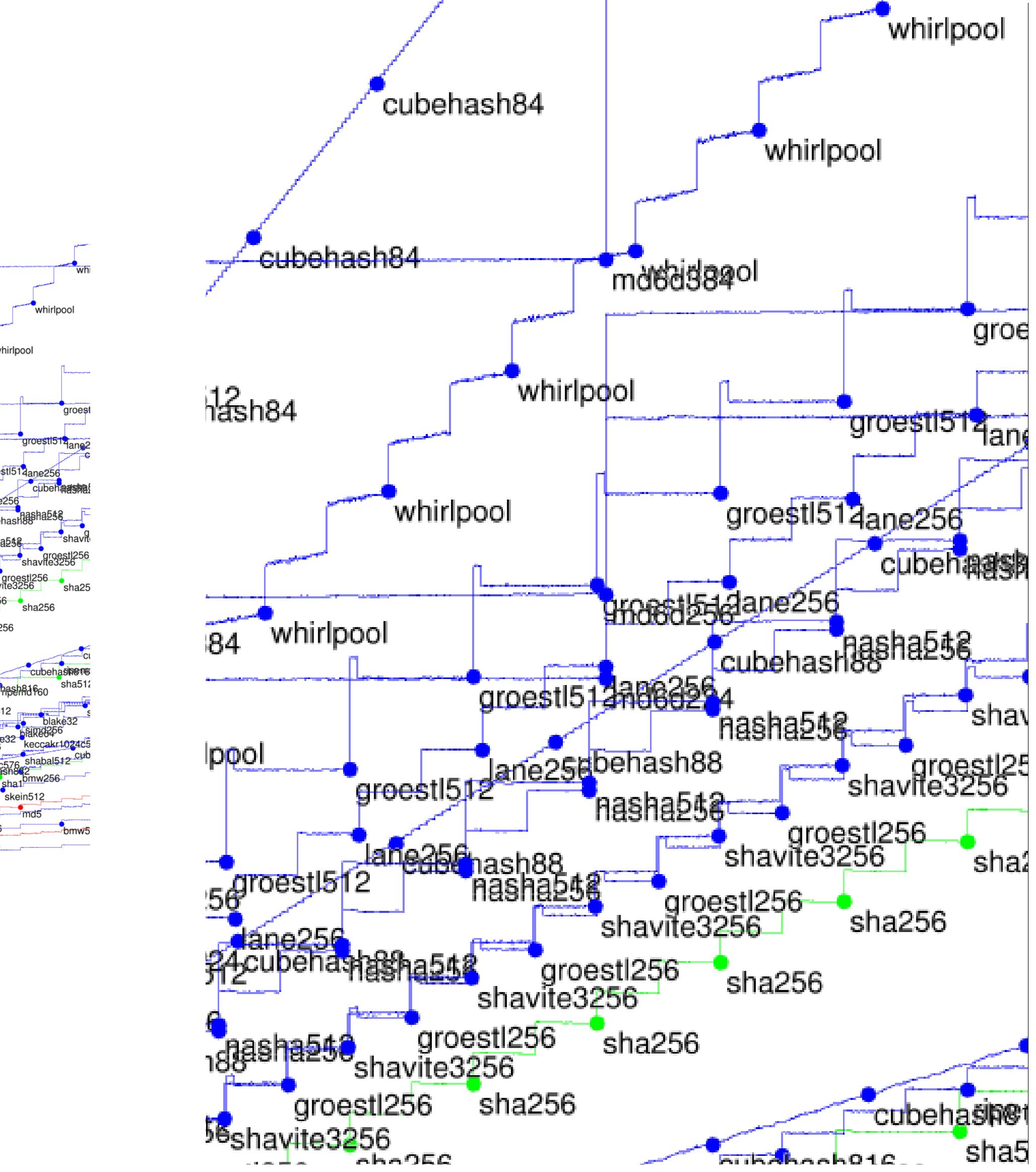
```
#define CRYPT
```

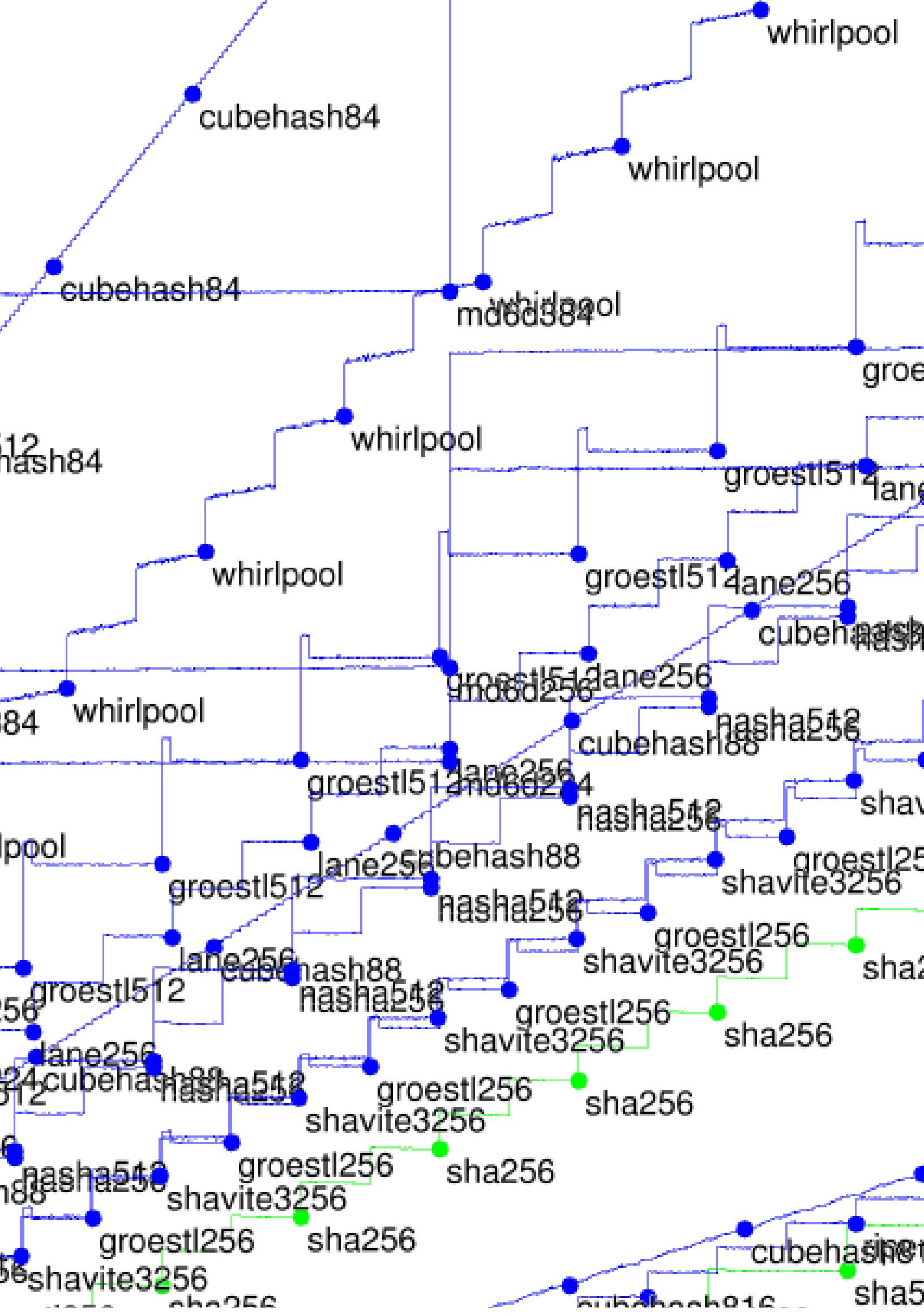


Submitter → eBASH

Define output size in api.

```
#define CRYPTO_BYTES
```

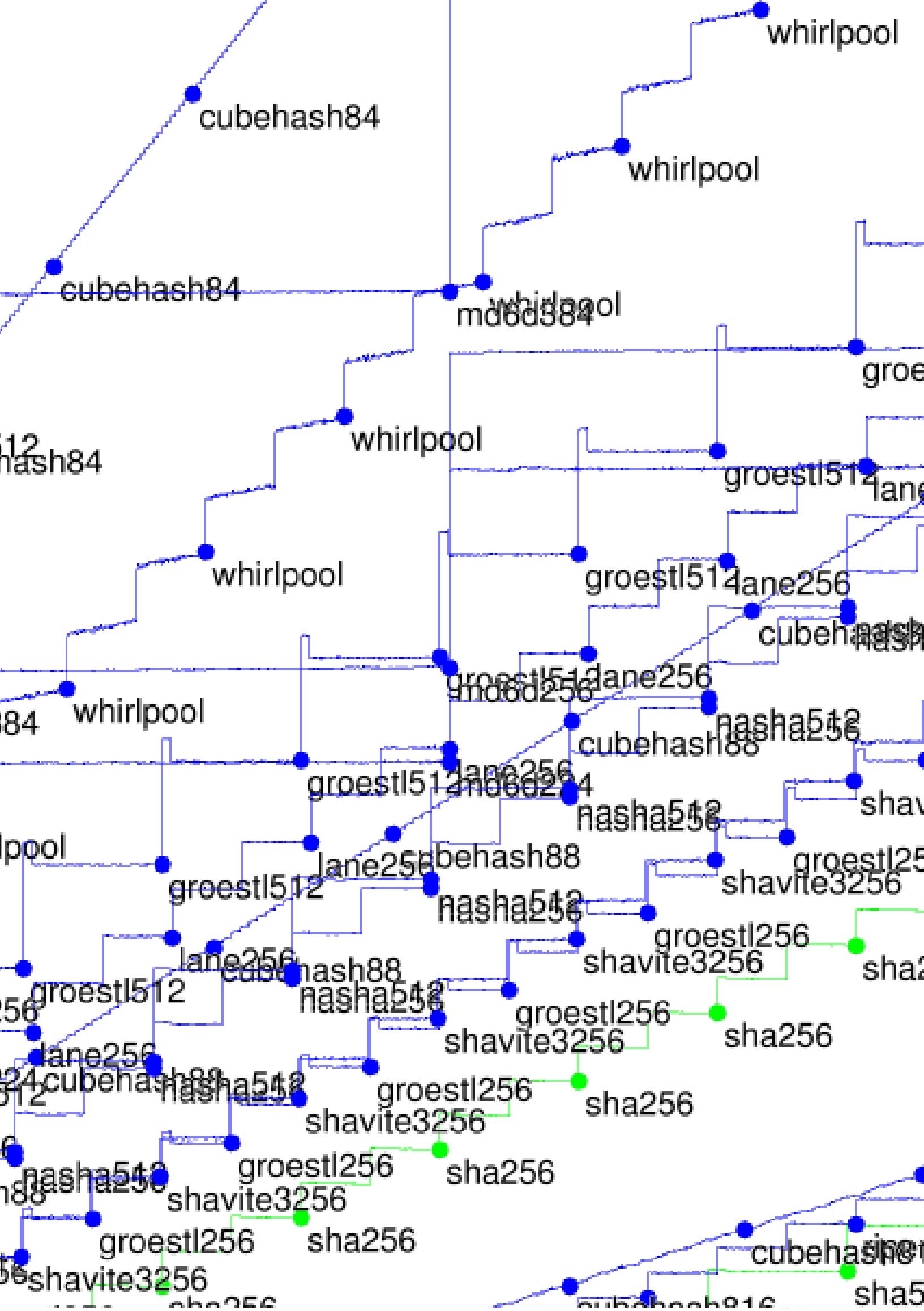




Submitter → eBASH

Define output size in api.h:

```
#define CRYPTO_BYTES 64
```



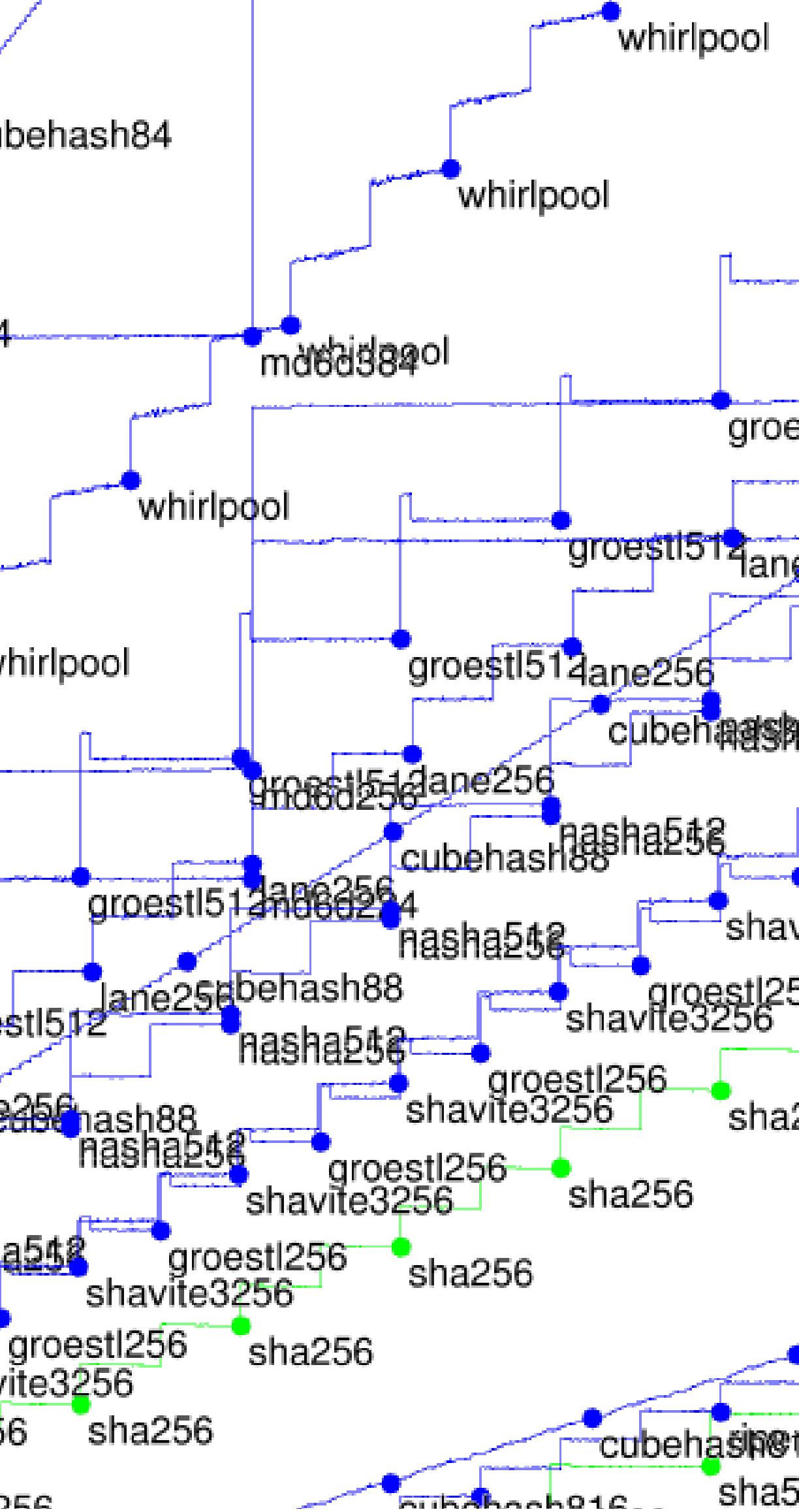
Submitter → eBASH

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:

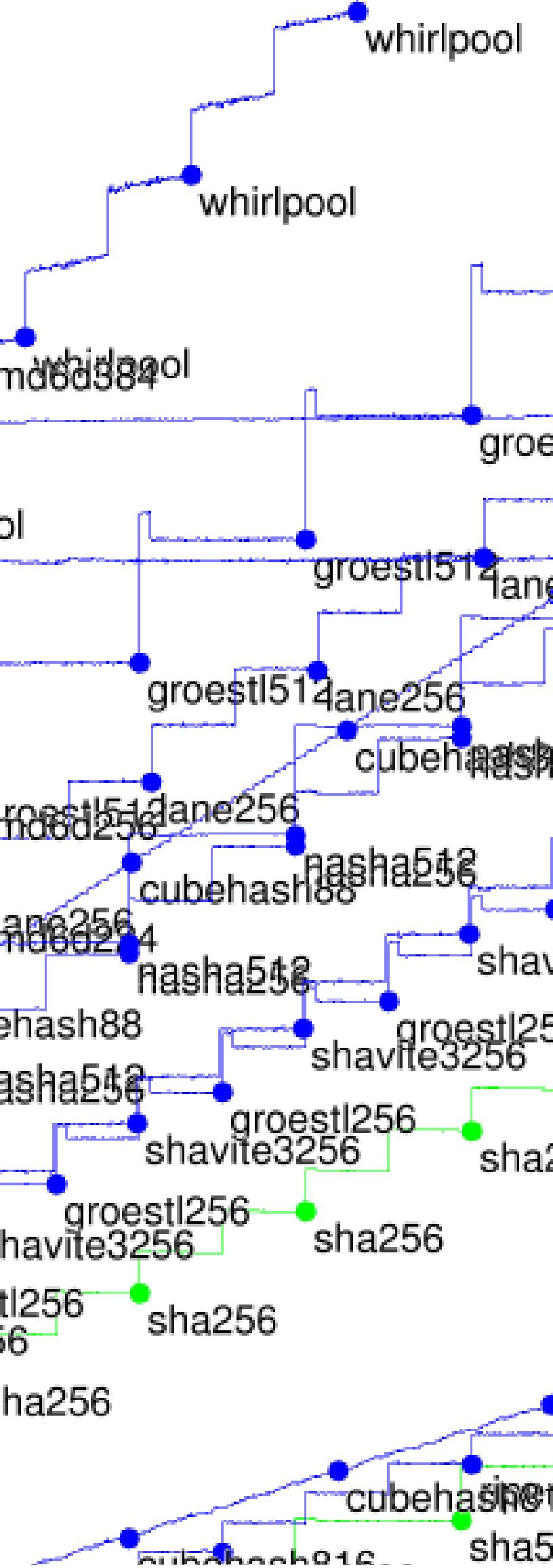
```
#include "crypto_hash.h"
#include "SHA3api_ref.h"
int crypto_hash(
    unsigned char *out,
    const unsigned char *in,
    unsigned long long inlen)
{ Hash(crypto_hash_BYTES*8
        ,in,inlen*8,out);
return 0; }
```

Send to
the URL
with one
crypto
contains

Measure
Much easier
to do you

More details
<http://callisto.csail.mit.edu>

Also easy
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 tar
with one directory
crypto_hash/your
containing hash.c

Measurements m
Much easier than
to do your own b

More details and
<http://bench.eBASH.org/call-hash.html>

Also easy for thi
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/:
containing hash.c etc.

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

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](http://bench.cr.yp.to/call-hash.html)

Also easy for third parties
to run the benchmark suite.