Micromint Extensions

David Wheeler

Notes August 1996
Revised November 1996

Introduction

Rivest and Shamir's Micromint can be enhanced while retaining most of the desirable properties. The approach is to use a sequence of hashed values for a transaction. A forger has to generate hashed values to a comparable density and so perform a similar amount of work to the legitimate generator. The work ratios are similar to or better than Micromint.

Basic ideas.

We consider the same model as Rivest and Shamir [] with three components. bank (broker), vendor, purse (user). The bank generates a large number of hashed values so the values are uniformly possible across the value field. The chance of each point having a value is identical. The values could be $h(x) + 2^{-t}x$ where x is a t bit integer, or just the quantised h(x).

We now take some sequential hashed values which are necessary for a transaction and test the density of hashed values. If the number of samples is large the density is determined to a reasonable precision and the forger has to create almost as many samples as the bank.

Each purse is issued with a sequence of hash arguments such that the hash values are monotonic and just above a known sequence of values, such as 64m+offset. The purse may be given the offset. The "purse" holds the arguments, a pointer to the first unspent argument and the number remaining unspent.

The coins are spent in sequence, from a purse to a vendor and a transaction will be the transfer of a number of hash arguments. The range of hash values usually represents the transaction value. For some uses the number whose hash gives the offset and purse number may be sent to the vendor in addition to any coin numbers.

The vendor will send the hash arguments to the bank, having performed appropriate checks.

Not all the hash arguments of the sequence will be sent from the purse to the vendor [suggested by Mark Lomas] and some will be selected by a formula, probably a simple hash, using a bank vendor key which is shared by the vendor and the bank, the transaction amount, and number of the first coin of the transaction. The vendor does the selection. Thus if a vendor tries to spend the coins with another vendor rather than or in addition to sending them to the bank, he will be unable to supply some of the coins requested and there is high chance of immediate detection by the receiving vendor.

By choosing a large enough step say 64 or above, the vendor can check the purse has complied with his selection with very high probability. Different purses will have different offsets and their values will interleave.

An additional reason for a large step is that errors of successive values in the sequence become totally uncorrelated, This allows a simpler mathematical model and tighter limits to be set. Otherwise consider the effect of say a gap of 10 in the hash sequence. This would affect the sum by by about 9*8/2=36 causing the limits to be much weaker.

Failures

The is always a chance that there will be a failure to meet the criteria due to chance. We can keep this chance small by using criteria with a very small probability of failure. Unfortunately this makes it slightly easier for the forger to satisfy the criteria. See the table below.

Forger's work ratio table

n	bank sum limit	yield 10^{-1}	$^{-3}$	$^{-6}$	$^{-9}$
5	32	13	46	240	-
10	42	7	15	35	84
20	60	4	7	11	18
40	90	3	4	5	7

Gives ratio of bank's work to forger's work to obtain various yields when the bank's chance of a failure is 10^{-9}

For large numbers of coins there is almost no problem as the loss in going to very strict criteria is very small. For smaller number of coins we can pretest each sequence and reject those which exceed some tighter criteria. A slightly tedious program is needed if this approach is pursued.

We can relax the limit to weaken the criteria, or we can just accept that we will get occasional faults. On-line to the bank they will cause no trouble, and off-line we can do what is normal, proffer more money and refer to the bank the sequence of unacceptable coins. On-line the bank can easily check and authorise the payment, off-line it can credit the purse when it is recharged.

Using a hashed sequence.

For each use of a sequence we need to have a minimum number of coins used to make forgery difficult. Thus if each coin is a cent we may have a minimum use of say 8 coins, and to allow vendor selection a range of perhaps 10. Thus the simplest solution would be to have a minimum transaction of a dime.

This may make some transactions difficult. We can code the transaction and choose to represent the values 1-10 by some of the 28 coded values, assuming we use the first, last, and 6 out of the remaining 8 numbers. This breaks the connection between range and value and requires that the bank accounts correctly. The value in a purse range is diminished by a factor up to ten when all transactions are of one cent. For transactions larger than a dime there is no change.

Another solution is to make transactions less than a dime probablistic. [TUB] For example, instead of a charge of 4c give 10 cents 4/10th of the time. This will eliminate the extra book keeping. The chance can made by combination of the first coin, vendors number choice and coin numbers selected by the vendor so that a purse, arbitrator or the bank can verify the transaction.

If either of the latter approaches are taken the value of a coin may be increased, by a little extra coding. If a coin is ten cents, we can use the combination of the first numbers to give a value from 1-10 and the rest to represent 10 cent, else we can code probablistically values 1-100 in the first 10 coins and let the later numbers of a transaction represent dimes.

In the latter case the relation between length and value is almost retained for the purse. The transaction may be left on a probablistic basis for the purse, but the bank can work out the details and transfer the correct amount. This does give extra work for the bank and extra numbers have to be transferred from the vendor to the bank. However, the vendor may accept the averaging, and reduce the bank vendor traffic (unless called upon to do so). The most desirable solution would be for the purse to accept the averaging, although that may be more difficult to sell.

Unfortunately, reducing the number of coins reduces the space and work needed by the forger as well as the bank. The work levels can easily be restored by choosing a slower hashing function, but the space needed is still reduced.

It is easy to extend the protocol so that repeated spending of a coin is catered for and payment for telephone calls or repeated selection of pages from an on-line catalogue can be catered for in an on-line kind of way.

Hashing Requirements

The hashes have to depend on a large parameter so that the forger cannot work out hashes in advance. There are constraints used in the number to be hashed which can be issued daily, but all are applied to the generation. The latter have to increase the length of the number used to hash. If assume 10 coins and the chance of a constraint detecting a forgery to be 1/2, then a constraint has to eliminate about 1/14th of the hash indices and we require about four extra bits to do one month's constraints. The larger transactions can detect forgeries with a higher probability.

These constraints can be applied before the hashing so the usage of the hash values remains close to 100%.

Checks

In cases where the purses coins issue hash values monotonically, the bank records for on-line access consist of just two numbers per purse (or even one). Thus checking on-line for used coins or forged coins could be communication limited.. A stronger check for a transaction would verify that each number of a transaction was issued and not already spent.

Forgery can be detected because the set of numbers created by hashing should be different for the forger and bank. Thus some will not have been issued. Coin replication by the buyer cannot be prevented except by tamper proof purses, but the duplicated coins can be tracked to the purse after the event, and the vendors will be traced by the bank. A list of purses which duplicate coins might be circulated so that a further use might be detected by some of the vendors and need on-line verification.

A purse could have an identifying number as a slight help against forgery. A hash identity number could contain the exact offset, purse number, and some known digits. This enhances each transaction by the sequence being longer, allows single coin spending with rather weak security, and simplifies the descriptions and calculations. Otherwise we need to compute the minimum of a number of coins, and essentially reduce the security to that of n-1.

An illustrative design.

The bank generates the hash values at unit density complying with all the constraints.
The bank issues each purse with arguments to a range of hashed values corresponding to the money needed.
The bank issues the hash parameters to all the vendors and individual vendor keys to each

vendor.
There is a time limit, say five weeks for the purses validity after which the coins require on-line authorisation from the bank.

Purse Vendor transactions

After mutual agreement about the price. The purse sends to the vendor the first and last number of the range corresponding to the amount and the number for the purse and offset.
The vendor issues a coin choice.
The coin choice is a hash of vendor key, transaction amount, first coin number from purse. The purse sends the corresponding coin numbers.
The vendor applies hashes and checks each number has a positive error less than 64.
The vendor checks the computed sum is less than f(s), a function of the number of samples.
The vendor statistically asks for on-line checks with the bank.
The vendor applies the constraint checks given by the bank to each entry.

Vendor Bank transactions.
The bank sends a new constraint daily.
The value transferred to the vendor is the virtual range.
The vendor transfers some or all of the coin numbers to the bank.

The bank marks “spent” all the coins in the transaction range.
The bank applies its own extra checks for forged coins or coins used more than once.
Some numbers are used by more than one purse, but the bank has enough information to detect false duplicates.
The bank credits the vendor with the transaction value.

At the end of a suitable time the purse is taken to a bank terminal for crediting and reloading. Further checks can be applied.

There are many arbitrary decisions and various tuning parameters are not defined, such as size of coin, role of betting, which checks to perform, amount of on-line checking, actions to be taken when forgery or duplication is detected and actions required if a transaction is aborted.

All aborted transactions are reported to the bank and an adjudicator. The bank can make immediate decisions but the secret data can be made over to the adjudicator at the coin expiry time, and a late confirming? decision made by the arbitrator.

Characteristics

Only one hash value is generated per coin.
Longer coin sequences more secure than small ones.
Purses merely supply numbers so almost no control is needed.
The bank generation is merely posting hash entries to memory after applying all constraints.
A purse can be replicated unless tamper proof, but double spending is detected at the bank.
A vender cannot spend the coins he receives without a high chance of getting detected by the receiving vendor immediately.
The bank will detect at least half the forged coins by its own private constraints. Thus the

chance of a group of 8 numbers escaping detection is less than 2^{-8}
The vendor will use the daily released constraints to immediately detect forgeries which have not used these constraints.
The purse requires storage for each coin that it holds, perhaps 48 bits per coin.
The purse vendor communication transfers only a fraction of the coins used in each transaction, except for very small transactions.
The vendor bank traffic may be even less, particularly if the vendor is required to retain the information in case the bank needs it.
Work per coin generated.
We require to apply constraints to a range and expand the number of bits at each constraint. The argument is then placed in the memory at the hash location. The sequential values for purse filling can be read off very easily. The bank has to check returned values against stored values, and mark ranges as being spent.
Filling purses requires transfers of large amounts of data, perhaps 48 bits per coin.
Each transaction involves a purse transferring some of the coin numbers to a vendor. The vendor needs one selection hash, one hash for each coin transferred, constraint checks for each coin transferred, and sending the first and last coin of each transaction to the bank, and more if the bank requests it.

Notary Service

An interesting notary service can be provided. A document is combined with a set of selected coins from a purse or bank. The hash of the document corresponds to the selected coins.

The hash function on each of the hash coins will give a simple check on the signature, which can be forged with a considerable amount of work. A check by the bank would confirm the coins used obeyed its private constraint and that no forgery had taken place.

The number to issue is adjustable. We could split the hash, say 160 bits into say 40 numbers from 1-16 and select coins at distances of 1 to 16 units. This would use a range about 17/2 *40 units (64?) from the purse.

If a cheapskate used the range again for a different document the probabilities are worse but not by much. Many uses of the same range would make it easier for a forger.

Comparison with Micromint

The idea of sequencing can be applied to the original Micromint. We can utilise the extra security of a sequence of hashed values each having a fourfold coincidence. We can also utilise the vendor selection idea. However, using the sequences directly seems more flexible.

If we use a unit of 1/4 cent, we require about the same effort as in Micromint, however as the margin against fraud can increase with the size of transaction, far fewer hash numbers and a larger unit can be accommodated for the same degree of safety.

For larger transactions, we need to transfer a small fraction of the total numbers used in that transaction while the same level of security is maintained.

The comunnication cost can be kept lower than for Micromint.

It is possible to use a coin of higher value than 1/4 cent. We could use 10 cents a unit and probability for the remnants of transactions which are not multiples of 10.. However, this would reduce the bank and forgers work load and space needed by a factor of 40. It is easy to make the hash slower, but to keep the memory requirements we can simply generate 40 times as many coins but issue only 1/40th of them. The forger can reduce his memory requirements by storing a fraction say 1/100th of the coins but at the expense of 100 times as much work per coin he produces.

References

Appendices

Formulae

Given a hash density 1/d. For a hash value of t bits, this will mean generating $2^t/d$ values. Given an arbitrary value, the chance of an error x in choosing the next highest value from the set of hashed values is $(1/d)e^{-x/d}$.
The mean error is d and it is positive.
Its standard deviation is d.
Truncating gives a bias of 1/2 and increases the standard deviation by a factor $\sqrt{1+1/3d}$, about 1.15 when d=1

The error y is defined as distance to the nearest higher value of the hash from a given value.
$y = \int_y^\infty e^{-x/d}.dx/d$, $y < 14d$ suffices to make the probability of an error $> 14d$ less than 10^{-6}

Simplifying by setting d=1.
The probability that the sum of m errors is less than y is given by
$\int_0^y (x^{m-1}/(m-1)!)e^{-x}dx$
$= e^{-y}\sum_m^\infty y^s/s!$ and may be replaced by zero with little degradation.

The probability that the sum of m errors exceeds y is
$\int_y^\infty (x^{m-1}/(m-1)!)e^{-x}dx$, the incomplete gamma function
$= e^{-y}\sum_0^{m-1} y^s/s!$
$\approx (1/y)exp((y-m+1)^2/2(m-1))$ for large m

If we define the error with respect to the sequence and minimum value we replace m-1 by m-2 in the above formulae.

Vendor calculations.

The vendor will receive a group of arguments for the current hash function. We first check that the asked for sequence has been sent. As the differences will be approximately multiples of 64, this check is easy. We now subtract the first (lowest) from all the group, removing appropriate multiples of 64, and checking for the minimum. We sum the group and subtract the product n*minimum. This is the sum to be checked. If the offset is part of the transaction, there is no need to compute the minimum and subtract the product.

If the bank uses as the virtual sequence m.64+offset+.5, with hash values extended by the argument, then with little loss of precision, the calculation can be done with the hashed values, and the rounding will have little effect.

Of course the calculation can be done exactly with extended hash values with little extra effort.

Constraints

The design of constraints requires care. They should be easy to implement and unbreakable even given millions of samples satisfying those constraints. However, the effective key length can be quite large, limited only by cache size.

They can all be implemented before the bank hash generation step so that no more bank hashes are needed but the forgers effort will need to be greater. He needs to generate sufficient hash values in less than a day to satisfy the fresh daily constraint, or else generate twice as many values the day before and then filter with the new constraint. To filter for m full constraints requires generation of 2^m as many values.

However, if he can determine the constraints from sample which satisfy them then his work needed is of the same order as the bank which perhaps used a month to generate them.

The constraints can be used to generate one extra bit at a time. For example to generate one extra bit from known bits, we can write a parity equation for the extra bit, but this is may be too weak. A slightly better way is to use three masks and find three parities add them and use the 2's sum bit. This has a much longer key to search.

We can use the operation a few times as follows. Generate a few bits with separate keys as above, say 5. Use them to select one of 32 keys to generate the required bit. This will not take minimal effort, but more likely to be secure.

Another use of the same idea would be to use partial constraints. Issue the keys to generate say one bit every 4 days. Then select one of 8 keys to give the constraint. The 8 keys are issued at the rate of 2 per day. Each constraint is independent but only one is used. Thus if numbers were generated but not satisfying the latest constraint, about one in sixteen would be detected. This will force the forger to generate more coins, and then use the latest constraint.

We automatically have an extra bank constraint as not all generated numbers will be issued to purses.

Attacks

1) Replicated purses.
The bank will detect the first coins spent twice, and issue a purse warning to vendors. Some vendors may delay their reports and take the consequences. Where double spending happens it should be reported to an auditor and arbitrator. They will adjudicate on which vendor to pay.

When the purse is returned to the bank for refilling, it may have extra information to resolve conflicts. It should certainly indicate which coins where transferred on each transaction. A search on that data for all vendors would indicate possible vendors, and the more numbers transferred the greater the resolution.

2) Stolen Purses.
Additional precautions of the usual kinds such as pin numbers fingerprints etc. can be taken to minimise the risk.

3) Vendor spends coins received.
The receiving vendor and coin selection will detect this almost every time.

A group of vendors combine their coins and spend the coins with another vendor. There is only a small chance of this happening, particularly if the purses are distributed randomly. Even so, it will require a large set to be combined to spend large amounts. If this is done on a large scale, then the bank will be able to gather data to identify the group. The group would have to be quite large before a transaction can be guaranteed not to fail, particularly a reasonable size transaction.

4) Forger generates coins.
If he is able to construct coins at the required density of a purse, and which satisfy all vendor known constraints then an off-line transaction will succeed. It will be detected by the bank (with high probability) as soon as the transaction is reported. It will usually be detected as a

forgery, but may be detected as a double spent coin. He may use a different purse sequence for each transaction, so that reporting invalid purses may not be much help. The automatic onlining of some transactions will give vendor detection.

It will be easier to forge coins for small transactions, but difficult for the forger to get a high return on his investment. The sum limit function can be tightened during a month so the compromise between sum failures and forgers effort is changed.

5) Forger gets or works out all the bank's keyed constraints.
Bank detects non issued coins, double spending, purse spending out of sequence, etc. and decides when to close down and refill purses.

Purse design

Its main function is to store many numbers, and be able to transfer (to a power source) selected numbers and hold the current spent limit and the final limit. A little microcode will be needed for protocol enforcement and ensuring that too many coins are not selected by the vendor. It may have a display giving the value left, value transferred etc., and it may have a commit or abort button, although these could be provided by the vendor.

Additional facilities would be to replace coins by transaction details, and preferably destroy or overwrite coins already spent. A tamper proof container would be useful.

Hacker Mentality

Many hackers would take such a system as a challenge, and set local unused computers to act as forgers. It is probably difficult to make much money that way without being detected.

Table giving the error sum limits y/(n-1), for some values of p
p is the bank failure rate.

n/p	10^{-1}	10^{-2}	10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}
2	2.30	4.60	6.91	13.82	20.72	27.63	34.54
3	1.94	3.32	4.62	8.34	11.97	15.55	19.10
4	1.77	2.80	3.74	6.38	8.89	11.35	13.78
5	1.67	2.51	3.27	5.34	7.29	9.18	11.04
6	1.60	2.32	2.96	4.69	6.29	7.85	9.37
7	1.55	2.18	2.74	4.23	5.61	6.93	8.23
8	1.50	2.08	2.58	3.90	5.11	6.27	7.39
9	1.47	2.00	2.45	3.64	4.73	5.76	6.76
10	1.44	1.93	2.35	3.44	4.42	5.36	6.26
20	1.30	1.61	1.86	2.49	3.04	3.54	4.03
40	1.21	1.41	1.57	1.95	2.28	2.57	2.84
80	1.15	1.28	1.38	1.63	1.83	2.01	2.17
160	1.10	1.19	1.26	1.42	1.55	1.66	1.77

The sum limit ratio y/(n-1) is the ratio that the sum limit has to be increased over the value needed if n is infinite, for selected probabilities of failure.

Table giving the forgers work ratio, y/(n-1), for some values of p
p is the usable fraction of those generated
Exponents in the table indicate scaling by powers of 1/10

n/p	10^{-1}	10^{-2}	10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}
2	0.105	0.010	1.00^{3}	1.00^{6}	1.00^{9}	1.00^{12}	1.00^{15}
3	0.266	0.074	0.023	7.07^{4}	2.24^{5}	7.07^{7}	2.24^{8}
4	0.367	0.145	0.064	6.09^{3}	6.06^{4}	6.06^{5}	6.06^{6}
5	0.436	0.206	0.107	0.018	3.12^{3}	5.54^{4}	9.84^{5}
6	0.486	0.256	0, 148	0.034	8.32^{3}	2.08^{3}	5.21^{4}
7	0.525	0.298	0.184	0.052	0.016	5.01^{3}	1.58^{3}
8	0.556	0.333	0.217	0.071	0.026	9.40^{3}	3.49^{3}
9	0.582	0.363	0.246	0.091	0.036	0.015	6.31^{3}
10	0.604	0.390	0.273	0.110	0.048	0.022	0.010
20	0.720	0.545	0.437	0.257	0.164	0.108	0.073
40	0.801	0.666	0.577	0.414	0.315	0.247	0.197
80	0.859	0.757	0.688	0.553	0.465	0.400	0.348
160	0.900	0.825	0.773	0.667	0.595	0.539	0.494

Limit tables for vendors.

These tables use quantised values for the initial probabilities. The mean and standard deviation are given, for sum of n values. The sum limit for various probabilities are given. The truncated tables just use the hash value and are biased. The rounded tables use hash values extended by the hash argument and rounded to the nearest integral value. The high limit is used to determine the banks limit while the low limit (at less dense hash values) is used to determine the forgers work requirements.

hash density 1 rounded high limit

n	mean	s.d.	10^{-1}	$^{-2}$	$^{-3}$	$^{-4}$	$^{-5}$	$^{-6}$	$^{-9}$	$^{-12}$	$^{-15}$
1	0.96	1.08	2	5	7	9	12	14	21	28	35
2	1.92	1.52	4	7	9	12	14	17	24	31	38
3	2.88	1.86	5	8	11	14	17	19	27	34	41
4	3.84	2.15	7	10	13	16	19	21	29	37	44
5	4.80	2.40	8	12	15	18	21	24	32	39	47
6	5.76	2.63	9	13	17	20	23	26	34	42	50
8	7.68	3.04	12	16	20	23	26	29	38	46	54
10	9.60	3.40	14	19	23	26	30	33	42	51	59
20	19.19	4.81	25	32	37	41	45	49	60	69	79
30	28.79	5.89	36	44	50	55	59	64	75	86	96
40	38.38	6.80	47	55	62	68	73	77	90	102	113
50	47.98	7.60	58	67	74	80	86	91	105	117	129
60	57.57	8.33	68	78	86	92	98	104	119	132	144
80	76.76	9.62	89	100	109	116	123	129	145	160	173
100	95.95	10.75	110	122	132	140	147	154	171	187	201

hash density 1 truncated high limit

n	mean	s.d.	10^{-1}	$^{-2}$	$^{-3}$	$^{-4}$	$^{-5}$	$^{-6}$	$^{-9}$	$^{-12}$	$^{-15}$
1	0.58	0.96	2	4	6	9	11	13	20	27	34
2	1.16	1.36	3	6	8	11	13	16	23	30	37
3	1.75	1.66	4	7	10	13	15	18	25	33	40
4	2.33	1.92	5	8	11	14	17	20	27	35	42
5	2.91	2.15	6	9	13	16	18	21	29	37	45
6	3.49	2.35	7	10	14	17	20	23	31	39	47
8	4.66	2.71	8	12	16	19	23	26	34	42	50
10	5.82	3.03	10	14	18	22	25	28	37	46	54
20	11.64	4.29	17	23	28	32	36	40	50	60	69
30	17.46	5.26	24	31	37	42	46	50	62	73	83
40	23.28	6.07	31	39	45	50	55	60	73	84	95
50	29.10	6.78	38	46	53	59	64	69	83	95	106
60	34.92	7.43	45	54	61	67	73	78	92	105	117
80	46.56	8.58	58	68	76	83	89	95	111	125	138
100	58.20	9.60	71	82	91	98	105	112	129	144	157

hash density 0.7070 rounded high limit

n	mean	s.d.	10^{-1}	$^{-2}$	$^{-3}$	$^{-4}$	$^{-5}$	$^{-6}$	$^{-9}$	$^{-12}$	$^{-15}$
1	1.39	1.47	3	7	10	13	16	20	29	39	49
2	2.77	2.08	6	9	13	17	20	24	34	44	54
3	4.16	2.55	8	12	16	20	23	27	38	48	59
4	5.54	2.94	9	14	19	23	26	30	41	52	63
5	6.93	3.29	11	16	21	25	29	33	45	56	66
6	8.31	3.60	13	19	23	28	32	36	48	59	70
8	11.08	4.16	17	23	28	33	37	41	54	65	77
10	13.85	4.65	20	27	32	37	42	46	59	71	83
20	27.71	6.58	36	45	52	58	64	69	84	98	111
30	41.56	8.05	52	62	70	77	84	90	107	122	136
40	55.42	9.30	68	79	88	96	103	110	128	144	160
50	69.27	10.40	83	95	105	114	121	129	148	166	182
60	83.12	11.39	98	112	122	131	140	147	168	186	203
80	110.83	13.15	128	143	155	166	175	183	206	226	245
100	138.54	14.70	158	175	188	199	209	218	243	265	285

hash density 0.25 rounded low limit

n	mean	s.d.	10^{-1}	$^{-2}$	$^{-3}$	$^{-4}$	$^{-5}$	$^{-6}$	$^{-9}$	$^{-12}$	$^{-15}$
1	3.99	4.02	0	0	0	0	0	0	0	0	0
2	7.98	5.69	2	0	0	0	0	0	0	0	0
3	11.97	6.96	4	2	0	0	0	0	0	0	0
4	15.96	8.04	7	3	1	0	0	0	0	0	0
5	19.95	8.99	10	5	3	1	0	0	0	0	0
6	23.94	9.85	12	7	4	2	1	0	0	0	0
8	31.92	11.37	18	11	7	5	3	2	0	0	0
10	39.90	12.71	25	16	11	8	6	4	1	0	0
20	79.79	17.98	58	44	35	29	24	20	12	7	3
30	119.69	22.02	92	74	63	54	47	42	29	21	14
40	159.58	25.43	128	106	92	81	73	66	49	38	29
50	199.48	28.43	164	139	123	110	100	91	72	57	47
60	239.38	31.14	200	173	154	140	128	119	96	79	66
80	319.17	35.96	274	241	219	202	188	176	147	126	109
100	398.96	40.21	348	311	286	266	250	235	202	176	156

hash density 0.25 truncated low limit

n	mean	s.d.	10^{-1}	$^{-2}$	$^{-3}$	$^{-4}$	$^{-5}$	$^{-6}$	$^{-9}$	$^{-12}$	$^{-15}$
1	3.52	3.99	0	0	0	0	0	0	0	0	0
2	7.04	5.64	1	0	0	0	0	0	0	0	0
3	10.56	6.91	3	0	0	0	0	0	0	0	0
4	14.08	7.98	5	2	0	0	0	0	0	0	0
5	17.60	8.92	7	3	1	0	0	0	0	0	0
6	21.12	9.77	10	4	2	0	0	0	0	0	0
8	28.17	11.28	15	8	4	2	1	0	0	0	0
10	35.21	12.62	20	12	7	4	2	1	0	0	0
20	70.42	17.84	49	35	26	21	16	12	5	1	0
30	105.62	21.85	79	61	49	41	34	29	17	10	5
40	140.83	25.23	109	88	74	64	55	48	33	22	15
50	176.04	28.21	141	116	100	88	78	70	51	37	27
60	211.25	30.90	173	145	127	113	102	92	70	54	42
80	281.66	35.68	237	205	183	166	152	140	112	92	75
100	352.08	39.90	302	265	240	221	204	191	158	133	113

hash density 0.1000 rounded low limit

n	mean	s.d.	10^{-1}	$^{-2}$	$^{-3}$	$^{-4}$	$^{-5}$	$^{-6}$	$^{-9}$	$^{-12}$	$^{-15}$
1	10.00	10.01	1	0	0	0	0	0	0	0	0
2	19.99	14.15	5	1	0	0	0	0	0	0	0
3	29.99	17.33	11	4	2	0	0	0	0	0	0
4	39.98	20.02	17	8	4	2	1	0	0	0	0
5	49.98	22.38	24	13	7	4	2	1	0	0	0
6	59.98	24.52	31	18	11	7	4	3	0	0	0
8	79.97	28.31	46	29	20	14	10	7	2	0	0
10	99.96	31.65	62	41	29	22	16	12	5	1	0
20	199.92	44.76	145	111	89	74	62	53	34	22	14
30	299.88	54.82	232	187	158	137	120	107	77	57	42
40	399.83	63.30	321	267	232	206	185	167	127	100	80
50	499.79	70.77	412	350	309	278	253	232	184	149	123
60	599.75	77.52	503	434	388	353	324	300	244	203	172
80	799.67	89.52	687	606	551	509	473	443	373	321	279
100	999.58	100.08	874	782	718	669	628	593	510	447	397

hash density 0.1000 truncated low limit

n	mean	s.d.	10^{-1}	$^{-2}$	$^{-3}$	$^{-4}$	$^{-5}$	$^{-6}$	$^{-9}$	$^{-12}$	$^{-15}$
1	9.51	10.00	1	0	0	0	0	0	0	0	0
2	19.02	14.14	4	1	0	0	0	0	0	0	0
3	28.52	17.31	10	3	1	0	0	0	0	0	0
4	38.03	19.99	15	6	2	1	0	0	0	0	0
5	47.54	22.35	22	10	5	2	1	0	0	0	0
6	57.05	24.48	29	15	8	4	2	1	0	0	0
8	76.07	28.27	43	25	16	10	6	4	0	0	0
10	95.08	31.61	57	36	25	17	12	8	2	0	0
20	190.17	44.70	135	101	80	65	53	44	25	14	7
30	285.25	54.75	218	173	144	123	106	93	63	44	30
40	380.33	63.22	302	248	213	187	166	148	109	82	62
50	475.42	70.68	387	326	285	254	229	208	160	126	101
60	570.50	77.43	474	405	359	324	296	272	216	175	144
80	760.67	89.41	648	567	513	470	435	405	335	283	242
100	950.83	99.96	825	733	670	621	580	545	463	400	350