POLAWIS

Arkadiusz Wysokinski, Ireneusz Sikora

15-03-2014

Cipher name: POLAWIS

Version number: v1

Designers: Arkadiusz Wysokinski, Ireneusz Sikora

Submitters: Arkadiusz Wysokinski, Ireneusz Sikora

Contact email address for submitters: polawis@sedkomp.com.pl

Date of the document: March 15, 2014

Specification:

- the basis for the calculation is conducted chosen prime p greater than 2^{256}, that is, for example, $p = 2^{256} + q$, where $q = 297$.
- portion of the input data (one block): 6 number of size 256 bits each, giving a 1536 bit
- portion of the output (one block of ciphertext):

  6 number of size 256 bits each, giving a 1536 bits, or

  6 number of size 8 * 256 bits each, giving a 12288 bits, in case you did not know how to perform calculations over the field Zp, and we need to conduct calculations on rational numbers Q (R); this happens with a probability of about $10 * q/p$ (theoretically) within the ciphertext, and always at the end of the ciphertext.
- key (exchanged between the communicating parties): 16 number of size 256 bits each, giving a 4096 bits
- additional data (associated data): 6 set of integers (each of size 256 bits, giving a 1536 bits); algorithm as default assumes all zeros, the parties may agree on a different string used when communicating (once or repeatedly), this sequence may not be confidential
- other conditions specified in the competition strings take size zero.

In details:

Introduction. We will present a method based on the properties of algebraic operations in noncommutative quaternion numerical field, allowing the replacement of the set six of real numbers, a set of six other real numbers designated with the two any real numbers.

Minimum number sequence that can be encoded is a string consisting of 6 numbers (if the string is shorter, it must be supplemented by selected constants). The next six numbers are treated as $a_1, a_2, a_3, a_4, b_2, b_3$ and b_4 we adopt as 0. We also adopt values for c_1 and c_4. If $a_4 \neq 0$ for such a set of numbers we determine the values of c_2 and c_3, and then x_1, x_2, x_3, x_4.

The encryption equations (for one round) are as follows:

$$
\begin{aligned}
&c_1 \in \mathbf{R} \setminus \{0\} \\
&c_4 \in \mathbf{R} \\
&c_2 = \frac{c_1(a_3 - b_3) + c_4(a_2 + b_2)}{a_4 + b_4} \\
&c_3 = \frac{c_4(a_3 + b_3) - c_1(a_2 - b_2)}{a_4 + b_4} \\
&x_1 = \frac{2a_1c_1 + c_2(a_2 + b_2) + c_3(a_3 + b_3) + c_4(a_4 + b_4)}{2(c_1^2 + c_2^2 + c_3^2 + c_4^2)} \\
&x_2 = \frac{a_2 + b_2}{2c_1} - \frac{c_2}{c_1} \cdot x_1 \\
&x_3 = \frac{a_3 + b_3}{2c_1} - \frac{c_3}{c_1} \cdot x_1 \\
&x_4 = \frac{a_4 + b_4}{2c_1} - \frac{c_4}{c_1} \cdot x_1
\end{aligned}
$$

The values $c_2, c_3, x_1, x_2, x_3, x_4$ were treated as *encoded*, and c_1 and c_4 were treated as *encryption key*.

The decryption equations (for one round) we be expressed as follows:

$$
\begin{aligned}
a_1 &= x_1c_1 - x_2c_2 - x_3c_3 - x_4c_4 \\
a_2 &= x_2c_1 + x_1c_2 - x_4c_3 + x_3c_4 \\
a_3 &= x_3c_1 + x_4c_2 + x_1c_3 - x_2c_4 \\
a_4 &= 2x_4c_1 + 2x_1c_4 - b_4 \\
b_2 &= x_2c_1 + x_1c_2 + x_4c_3 - x_3c_4 \\
b_3 &= x_3c_1 - x_4c_2 + x_1c_3 + x_2c_4
\end{aligned}
$$

It should be noted that the basic operation of encoding 6 real numbers (or integer multiples thereof) additionally needs only two real numbers. So in order to encode $n = 6m$ real numbers, we need $n + 2k$ real numbers (where $min(k) = 1$, and $max(k) = m$). without knowledge of c_1 and c_4 it is virtually impossible to guess the values of $a_1, a_2, a_3, a_4, b_2, b_3$.

Unfortunately, it is possible to perform here a type of attack - with a known-plaintext and ciphertext, as we have up to 6 equations and only two elements of the key. However, this drawback can be eliminated by using multi round version of the algorithm.

It turns out, that the values of a_4 and b_4 separately do not matter, what matters is the sum of $a_4 + b_4$.

If we assume that $b_4 \neq 0$ then the algorithm is somewhat strengthened.

So the basic operation of encoding (without zero) of 6 reals (or an integral multiple thereof) needs further only $(2 + 1) = 3$ real numbers. So to encode $n = 6m$ real numbers we need $n + 2k + l$ of real numbers (where $min(k) = 1$, and $max(k) = m$, and $min(l) = 1$, and $max(l) = m$). without

the knowledge of the values c_1 and c_4 (and additionally b_4) it is virtually impossible to guess the values of $a_1, a_2, a_3, a_4, b_2, b_3$. The constant b_4 may be included in the coding chip, but it has less impact on the ciphertext, so I will treat it as an additional element (perhaps explicit) of the algorithm. If we want to decrease the possibility of unauthorized decoding of the ciphertext we can recursively apply the basic equations presented earlier, that means repeat (n times) the basic encryption procedure without zero. Key variables in particular rounds can be constant (basic version – simpler) but it is better when for every round the key is different. Because of that we will need a key of $2n$ length, where n – number of rounds.

In every next round equations become more complicated. If for the measure of their complexity we assume the number of needed symbols to save them, then it appears that in the round $n+1$ we need around 6 times more symbols than in round n. It means, that after k rounds the number of needed symbols will be 6^{k-1} times larger compared to the first round. After eight rounds this coefficient will have the value of about $279,936$ (it is a lower estimate).

The important role here begins to play the approximation of intermediate result, so we may not get the exact results but the estimated ones. To get accurate results one can apply following conditions:

- as input data in the first round we put only integers and we choose constant b_4 so that $x_4 \neq 0$
- we round the results obtained in the last round.

In the practical application it is not an essential problem and it allows the application of multi round procedure operation with machine precision.

Proposed algorithm requires a larger number of calculations than in the one round version but the use of rewriting an earlier round output to the input of the following round causes the demand for CPU time to grow linearly with the increasing number of rounds and the direction number of this linear dependence is lesser than 1.

Proposed algorithm operates on real numbers. Program implementing this algorithm is based on the representations of real numbers available in *C* – *float* and *double* types. Because of too big rounding during computation I was made to accept the integer type (char) corresponding to writing number on the 8 bit (1 byte) as input data. Because of that a sequence coded in that implementation is 4 times longer than input sequence (float type = 4 bytes) for number of rounds up to 4 and 8 times longer than input sequence (double type = 8 bytes) for number of rounds from 5 up to 8. For longer number

of rounds double type is not enough and it is necessary to use a ciphertext which can save one real number on larger number of bytes than 8.

In the next rounds we got a lot of intermediate results which can be used to modify the value of coding key. For the first six data we get 6 new results after each round which are taken as input data for the following round. If we save those results "momentarily" then we can use some of them (exactly one number from each round) as new values for selected parts of the key.

So as for 8 rounds procedure we got 7 new values of key which we will use to the next six input data. Situation looks like follows:

- for the first six inputs elements we have 16 elements of the key (16 real numbers) but only 6 equations.
- for second set of six data numbers we have 7 new elements of key, which replaced 7 old elements (9 elements of key do not change) and of course there are 6 new equations
- for every next six input data situation repeats as for second set of six data numbers.

Those additional elements of key (7 for every new 6 input data) are nowhere saved nor sent, but escalated on the previous elements and input data, both on encoding and decoding side. So if length of input data (expressed by number of real numbers) is equal to $k = 6 * m$ then number of different values of key participating in coding and decoding is equal to $l = 7 * (m - 1) + 16$, where $m \in N$.

m	k	l
1	6	16
2	12	23
3	18	30
⋮	⋮	⋮

$$\lim_{m \to \infty} \frac{l}{k} = \frac{7}{6} > 1$$

Which means that for any m we have more elements of the key than equations. Additionally, only 16 beginning values of the key (only one-time sending of the key between communication sides or its transmission in other possible way required) are sent between the sender and the addressee.

Important feature of this algorithm is that lack of information about six previous elements does not let anyone to encode the further part of ciphertext so the condition for full decoding of ciphertext is its "integrity" (no

unauthorized modification of ciphertext, correct order of information in the ciphertext, no omissions – breaks - in the ciphertext).

Unfortunately there is an incidental consequence which makes it more difficult to do parallel computing which was quite easy in simpler versions - there every six numbers could be encoded and decoded separately. For even bigger complexity and randomness of the ciphertext it is suggested to use in applications realizing proposed algorithm a pseudo random number generator. The proposal of suitable working implementation will be presented in the end of this paper.

It is possible to use a modified algorithm in which where the operations are possible we calculate in a finite field and otherwise we calculate in rational numbers $\mathbf{Q}$ ($\mathbf{R}$). It significantly reduces the size of ciphertext as compared to the case when we calculate only in rational numbers $\mathbf{Q}$ ($\mathbf{R}$), unfortunately at the expense of increasing the computation time – looking for an inverse of a "modulo p" is much more complicated than "normal" division of real numbers.

Encryption procedure using calculations of finite fields. At the time of obtaining good results with the use of encryption calculations in qaternions field whose coordinates are real numbers (or rational), which is the number of fields belonging to the infinite (of uncountable or countable number of elements), it becomes natural to ask whether the encryption is analogous procedure used in place of one float field known finite fields.

The first finite, which decided to explore the possibility of using the field Z_p , that is, the field of integers with defined actions "modulo p", where p is a prime number.

Used equations (for one round) are the same as in previous algorithms, and so the encryption equations looks as follows:

$$
\begin{aligned}
&c_1 \in \mathbf{Zp} \setminus \{0\} \\
&c_4 \in \mathbf{Zp} \\
&c_2 = \frac{c_1(a_3 - b_3) + c_4(a_2 + b_2)}{a_4 + b_4} && mod \quad p \\
&c_3 = \frac{c_4(a_3 + b_3) - c_1(a_2 - b_2)}{a_4 + b_4} && mod \quad p \\
&x_1 = \frac{2a_1c_1 + c_2(a_2 + b_2) + c_3(a_3 + b_3) + c_4(a_4 + b_4)}{2(c_1^2 + c_2^2 + c_3^2 + c_4^2)} && mod \quad p \\
&x_2 = (\frac{a_2 + b_2}{2c_1} - \frac{c_2}{c_1} \cdot x_1) && mod \quad p \\
&x_3 = (\frac{a_3 + b_3}{2c_1} - \frac{c_3}{c_1} \cdot x_1) && mod \quad p \\
&x_4 = (\frac{a_4 + b_4}{2c_1} - \frac{c_4}{c_1} \cdot x_1) && mod \quad p
\end{aligned}
$$

and decrypting equations (for one round) can be written as follows:

$$
\begin{aligned}
a_1 &= (x_1c_1 - x_2c_2 - x_3c_3 - x_4c_4) && mod \quad p \\
a_2 &= (x_2c_1 + x_1c_2 - x_4c_3 + x_3c_4) && mod \quad p \\
a_3 &= (x_3c_1 + x_4c_2 + x_1c_3 - x_2c_4) && mod \quad p \\
a_4 &= (2x_4c_1 + 2x_1c_4 - b_4) && mod \quad p \\
b_2 &= (x_2c_1 + x_1c_2 + x_4c_3 - x_3c_4) && mod \quad p \\
b_3 &= (x_3c_1 - x_4c_2 + x_1c_3 + x_2c_4) && mod \quad p
\end{aligned}
$$

Unfortunately, it is not possible simply replacing of the calculation in real numbers by calculations in the field Z_p. This is due to the fact that some of the calculations that are naturally feasible in the field of real numbers are not properly enforceable in the field of integers modulo p.

The problem relates only to the equations corresponding to the coding, as occurs in operation of these equations division. As we know, in this field the numerical division by "zero" is not feasible. For calculations in real enough so appropriate selection b_4 constant, so that $a_4 - b_4 \neq 0$, which was possible to implement globally for all possible sets of input data, and to provide the appropriate selection key (the key to the reduction of non-zero values corresponding to variables c_{1n}).

For the calculation of the field Z_p the above two conditions (a condition wherein the first solid b_4 can not be selected globally) must be added another:

- $2 * c_1 \neq 0$ - there is not enough only to $c_1 \neq 0$, you also exclude the value of $c_1 = p/2$, since $2 * p/2 = p = 0 \bmod p$. Unfortunately, it is difficult to make off with the method another key value, especially

when we use a variable in subsequent rounds, and for the next batch key data as what values should be excluded depends on the input.

- $2 * (c_1^2 + c_2^2 + c_3^2 + c_4^2) \neq 0$ - also there is not enough only to $c_1 \neq 0$, but the value of zero can occur for a number of sets of non-zero-values of variables $c_1, ..., c_4$. He talks about the hypothesis Waring (proven by Hilbert) in the part called the Lagrange theorem, which states that every non-negative integer can be expressed as the sum of four squares of integers and such presentation is ambiguous. For us it is important that it be considered an expression can have values that are integral multiples of p, that the calculations "modulo p" is zero.

- where $p > 2^k$ (k-number of bits in a processed word) there was a problem with the transmission message on the transmission channel value between $\langle 2^k, p)$, theoretically possible to increase the length of time transmitted word, but it is a "low cost"

- where $p < 2^k$ (k-number of bits in a processed word) there is a problem with processing input data in the range $(p, 2^k\rangle$, and this problem is indelible.

- for each file there is a problem with the message EOF (end-of-file), for Linux (Unix) is -1 (the number is not representing any valid ASCII character, appearing in the file once at the end), but $-1 = p - 1$, and the number of $p - 1$ is a valid input variables, it is a problem indelible.

Thus, the direct application of the finite calculation is not possible because of some simple arithmetic will not be able to perform in a correct way for any input, and a sufficiently large space limit values for the key.

However, you can apply the modified algorithm in which where actions are feasible we calculate the finite field (Z_p), and otherwise we are counting in the field $\mathbf{Q}$ ($\mathbf{R}$).

This way you can adjust the calculation of the finite each of the previously mentioned algorithms for real numbers.

Here I will try to accurately describe the algorithm and the results offered by the use of the procedure eight-round bouts, the replacement of seven key elements in each round.

So you should start by deciding which transmit channel width (size in bits) we will use. The choices here are partly limited by the parameters of digital machines available and accessible programming languages. In the case of PC, Linux and C language will be selected as the optimal width of the base $4 * 8 = 32$ bits. Another possible width of the transmission channel is odd multiple of 32 bits, or 64, 96, 128, ... , 2048, ...

Next, determine the value of p possible nearest number 2 to the power of the "transmission channel width", which channel 32-bit looking number p close to 2^{32}, and generally for the channel k-bit looking for the number p close to 2^k. Of course, they found a prime number is closer to an appropriate power of two, the better.

Then, for the assumed value of p respectively generate "key" (keeping in mind that the key length also depends on the number of rounds of encryption and key contains two numbers for each round). When generating a key so we select the items to the value corresponding to c_1 and $2 * c_1$ were different from zero (and this is dependent on the choice of p).

Then we can, but this is optional, generate "zero head ciphertext" as follows: using the selected key encrypt input string consisting of all zeros (six numbers representing a value of zero in the selected "bit range"). As a result of this operation, we get the ciphertext, in which the positions corresponding to x_1 and x_4 get a value different from zero (depends on all the elements of the key), and other things will be zero. Decrypted the" zero" the head can immediately tell if the receiving party uses the key message wrong - any of the decrypted number is non-zero, or potentially valid. This may be required, for instance when the communicating parties have agreed to the use of multiple keys previously agreed, to quickly access the key. In general, it is better that part of the encryption process omitted, since it facilitates the process of "strength" break the cipher.

To ensure that the ciphertexts differ in case of multiple encryption of the same message M with the same key K before encrypting messages to the right to generate a unit serving (six numbers) random information and subject it to the process of encryption. These random six numbers will help us with a fixed key, to get the indirect calculation after each round of which will be used in the encryption process the next chunk of data (the next six numbers taken at the entrance). Exactly the same intermediate values is obtained in the process of decoding the random six (keeping in mind that they will appear in reverse order). In the proposed implementation of the algorithm for the adopted eight rounds of coding, I use seven intermediate values retrieved from the variable x_1. Yes received 7 new values so 7 replaces the old key elements (such as the values they can take the value zero, the substitution are key elements corresponding to the variable c_4).

Due to the *a priori* to guarantee the feasibility of calculations on real numbers above do with the version of the algorithm that operates on real numbers (rational).

Since at least once (to send EOF) will have to apply the algorithm in version "over the infinite field", the key will be treated in two ways: first, it will be loaded as a sequence of real numbers (floating point type - *float*),

for the second time as a sequence of integers. Therefore, two times encrypt and send a portion of random information for the first time to replace part of the key value read as real numbers that will be used for encryption and decryption "algorithm real", the second time to replace the key value read as integers, which will be used for encryption and decryption "algorithm integer". Random encryption information for the adjustment of the key completely can and should make a " integer algorithm", keeping in mind that when you generate the random pieces of information to ensure that it is properly given to encrypt (the fulfillment of all the conditions for the calculation of the finite field of integers modulo p) and thus decrypt.

With this preparation we can proceed with the proper encryption M. If the appropriate conditions are met - encrypt "integer algorithm". An important feature of this method of encryption is that the length of the ciphertext is identical to the length of the input data for a single serving (six integers) input.

In case of a read six numbers on the entry in the field of finite action are not feasible, we need for that, and only that portion of the information from the input "switch" to calculate the real numbers. In addition, we have the fact "switch" to notify the receiving party ciphertext. To this end, before submitting the ciphertext algorithm processed "real" we must send a string set that is "signal switch". The tests for this purpose used a constant string consisting of the characters with the number 254 in ASCII (FE - hexadecimal), the length of a single chunk of data in the "integer" algorithm. It seems that one can use any string constant, any different from the" zero", that is, do not use ASCII characters -" 0". This part of the ciphertext will have a size nine times the size of the input data (one time "string indicating" + 8 times "real" ciphertext).

Active repeat until you run out of data input.

The decryption process is much simpler - first "zero head decrypts", if present, then the algorithm reconstruct real random information in order to obtain the correct key, the real, then the "integer algorithm" random reproducing of information to obtain a complete correct key. Then, if we read "fixed string information", the decryption process for a single chunk of data encrypted algorithm we "real", and in the opposite case, we use the algorithm "integer".

Active repeat till the end of the encrypted data in the input, or decrypt EOF.

A trial implementation done as a program in C language with an attached library *gmp* (The GNU Multiple Precision Arithmetic Library) available under the GPL. The whole thing was compiled compiler *gcc* and runs on Linux. The demand of the software on working memory is low, on the order of sev-

eral hundred sizes used the number p. However, the demand for CPU time is very significant. The decryption process is about three times faster than the encryption process.

From the utility point of view, it is best if the largest part of the calculation will be carried out on integers. Border situation (optimal) is the switch to calculate the real number will be only once - to mark the end of the file transfer. This can be achieved by choosing sufficiently large p. It should also be noted that too high a value of p can cause a significant slowdown in the calculations - the need to process very large numbers.

To conduct the calculations for "normal" processors (ie 32 or 64 bits) seems to be the optimal use of the p from close 2^{64} to near 2^{512}.

It seems reasonable to apply specialized processors with bits increased in order to accelerate the calculations. Today, such processors are used in graphics cards and the appropriate software, they may be useful in the implementation of the" hardware" version of the encryption / decryption using quaternions.

The resulting ciphertext has the following properties:

- Humming normalized distance between the input string and the string of bits encrypted calculated for the values very close to 0.5 - which means that almost exactly half of the bits to be replaced (based algorithm for "integer"),

- Humming normalized distance (calculated for the "integer algorithm") between the input string and the encrypted string to byte counts (1 byte = 8 bits) takes values close to 1 - which means that almost all the characters are exchanged (but not exactly all) ,

- distribution of characters (ASCII) in ciphertext is close to the standing, i.e. the frequency of each character in ciphertext is similar, and the longer the ciphertext, the coefficient of ("incidence of the most common character" - "incidence mark rarest") / "incidence the most common character" sign is decreasing (for a file about 1.5 GB - less than 1%),

- ciphertext undergoes virtually no compression (checked publicly available programs - gzip - compression of the finite size of the dictionary, and bzip - compression with variable size of the dictionary),

- if we compare two ciphertexts generated for the same input data and at the same key, but in two different encryption processes is more than 95% characters (ASCII) is changed,

- factor not get worse if we change one character (ASCII) in the key,

- ”any error” in ciphertext misrepresentation or mistake when calculating the correct decryption whereas a message from a place where there was a calculation error or misrepresentation transmission.

The test program did not use replacement patterns, depending on the value of the additional key element - it gave only 256 (for 8 rounds) additional options, which is the number of very small compared to the possible number of different key values - for example, 32 bit * 16 = 512 bits , that is, 2^{512} - and this is the minimum value. Next possible to choose is 1024 bits, and actually there is 8192 bit - for p close to 2^{512}.

Security goals: That the cipher is designed to provide the maximum possible robustness, full integrity and confidentiality Each performed the encryption process outputs the other (different) ciphertext (even in case of multiple encryption of the same plaintext using the same key) - a difference of more than 95 % of ASCII characters.

parameter	number of bits of security
confidentiality for the plaintext	> 256
integrity for the plaintext	> 256
integrity for the associated data	> 256

Security analysis: for initial processed block :
16 elements of initial key - 6 equations = 10 elements of key is a free parameters
In bits: 8 X 255 + 2 X 256 = 2552 bits (8 elements of key is nonzero)
for each next processed block :
9 elements of key + 7 elements of key changing in each block- 6 equations = 10 elements of key is a free parameters
In bits: 8 X 255 + 2 X 256 = 2552 bits (8 elements of key is nonzero)

Features:

- ciphertext size is slightly larger than the size of the plaintext
- each performed the encryption process outputs the other (different) ciphertext (even in case of multiple encryption of the same plaintext using the same key) - a difference of more than 95 % of ASCII characters
- ensured high integrity ciphertext - change ciphertext fragment, adding anything to the ciphertext, or remove any part of it causes the inability to properly decrypt the rest of the message - that the lack of integrity

further confirmed using the "additional data" encrypted and transmitted at the end of the ciphertext

- exchanged between the parties 16 numeric key is the only key that initiates the encryption used in unchanged form only for the first 6 numeric block of input data, for each subsequent block of data that is changed 7 numbers in the primary key, so that the effective length of the used key is equal to $16 + 7/6*$ k, where k is the number of blocks for which the plaintext is divided, therefore, the effective length of the key is greater than the length of the plaintext

- the resulting ciphertext is obtained in the statistical tests recommended by NIST results comparable to those obtained using the ciphertexts: DES, 3DES, and AES; ciphertext exhibits the characteristics of white noise

- a test application has been successfully compiled on selected 32-bit systems (Debian 4) and 64-bit (Centos 6) in single-threaded mode, the use of multi-threading is possible, but requires a significant change in the algorithm

- application in non-optimized version requires a lot of CPU capacity, the other parameters of the equipment required is minimal and met with a supply of the currently available computers (also tested on the hardware over 10 years old)

- note! before it is desirable to encrypt the input data subjected to a process to reduce the redundancy (e.g., by compressing them widely available programs)

the following relations are satisfied:

$$\liminf_{n\to\infty} \frac{\|K\|}{\|M\|} = \frac{7}{6} > 1$$

$$\liminf_{n\to\infty} \frac{\|C\|}{\|M\|} = 1$$

$$\forall_{n<\infty} \frac{\|C\|}{\|M\|} > 1$$

$$\|C\| = n\|B\|$$

where n - numbers of blocks, $\|B\|$ -length of block, $\|C\|$ -length of cipher, $\|K\|$ -length of equivalent (effective) key, $\|M\|$ -length of message.

Design rationale: The project uses a new algebraic methods to obtain high-quality ciphertext with potentially high resistance to unauthorized access to encrypted information.
The designer/designers have not hidden any weaknesses in this cipher.

Intellectual property: The main part of the presented algorithm is the original (author's, not previously published) idea to apply to encrypt a transformation based on the algebra of quaternions.
Other tools and libraries needed to write test applications are available under the GNU licence.
If any of this information changes, the submitter/submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

Consent: *The submitter/submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitter/submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitter/submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitter/submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitter/submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitter/submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.*