ISO 9796-1 and the new forgery strategy

(Working Draft)

Don Coppersmith Shai Halevi Charanjit Jutla

IBM

August 23, 1999

Abstract

In this note we show how the new forgery strategy of Coron, Naccache and Stern can be modified to break the ISO 9796-1 standard for RSA and Rabin digital signatures.

1 ISO 9796-1 and the attack of Coron et al.

ISO 9796-1 [4] is a standard for RSA (and Rabin) signatures [7, 6]. In particular, it was designed to resist attacks that exploit the multiplicative structure underlying these cryptosystem. See [5] for a survey of these attacks, and [3] for the reasoning behind the ISO 9796-1 standard.

Recently, Coron, Naccache and Stern described in [1] a "new signature forgery strategy", which is a sophisticated variant of the Desmedt-Odlyzko multiplicative attack from [2]. In their paper, Coron et al. described an attack against a slight modification of ISO 9796-1, but this attack does not work "right out of the box" against the actual standard. In this note we show how a small modification to the technique in [1] can be applied to break the actual ISO 9796-1 standard. We start by quickly reviewing the format that was considered by Coron et al. and the attack against it, as well as the actual format used in ISO 9796-1.

1.1 The "new forgery strategy"

The ISO 9796-1 standard (and the variant the was considered by Coron at al.) specifies how a message m is encoded for a signature before applying the RSA operation to it. It uses a fixed nonlinear permutation $s(x)$ mapping 4 bits to 4 bits. Below we let $\bar{s}(x)$ be the result of setting the most significant bit of $s(x)$ to '1' (where x is a 4-bit nibble). That is, $\bar{s}(x) = 1000$ OR $s(x)$. The variant that was considered in [1] is as follows: Assume that the modulus length is $16z + 1$ bit (where z is even), and the message m is of length $8z$ bits. The message is encoded into a $16z$-bit integer by using an encoding function μ, which is defined as

$$\begin{aligned}\mu(m) \quad = \quad & \bar{s}(m_{\ell-1})\, s(m_{\ell-2})\, m_{\ell-1}\, m_{\ell-2} \\ & s(m_{\ell-3})\, s(m_{\ell-4})\, m_{\ell-3}\, m_{\ell-4} \\ & \ldots\end{aligned}$$

$$\begin{array}{l} s(m_3)\; s(m_2)\; m_3\; m_2 \\ s(m_1)\; s(m_0)\; m_0\; 6 \end{array}$$

where m_i is the i'th 4-bit nibble of m. To sign a message m, one needs to apply the RSA operation to $\mu(m)$.

The Coron et al. attack against this signature scheme proceeds roughly as follows. They consider 64-bit strings x of the form

$$\begin{array}{rcl} x & = & s(a_6)\; s(a_5)\; a_6\; a_5 \\ & & s(a_4)\; s(a_3)\; a_4\; a_3 \\ & & s(a_2)\; s(a_1)\; a_2\; a_1 \\ & & 2 \quad 2 \quad 6 \quad 6 \end{array}$$

where $a_6 \ldots a_1$ are any six nibbles, except that a_6 must be one of the eight nibbles for which $s(a_6)$ already has the most significant bit set to 1. Since x is short (only 64 bits), then there is a good probability that it will be smooth (i.e., will have only "small" prime factors – say all smaller than 2^{16}). Then they set

$$\Gamma = \sum_{i=0}^{z/2-1} 2^{64i}$$

and consider the $16z$-bit integers $M = \Gamma \cdot x$, which is just $z/2$ repetitions of the string x. Since x has the most significant bit set to 1 (because of the restriction on a_6) and the least significant nibble set to '6', then M is indeed a valid encoding of some message m. Namely, there exists a message m (which can be easily recovered from M) such that $\mu(m) = M$.

The procedure above is repeated many times, with different x'es, so as to generate many valid encodings $M_i = \Gamma \cdot x_i$ for which x_i is smooth. For example, if the smoothness bound that is considered is 2^{16}, then the attack needs to collect about 6500 such M_i's, since there are about 6500 primes smaller than 2^{16}. Once enough M_i's are collected, one can find "homomorphic dependencies" between these M_i's, and use these dependencies to devise a signature on one of these M_i's from the signatures on the others. See [1] for more details on the attack.

Remark (off-line work). An interesting feature of the attack from [1], is that essentially all the work is invested in finding the M_i's, and this work can be done off-line, before even seeing the RSA modulus. Once enough M_i's are collected, they can be used against *any RSA modulus* of the right length. The attack that we describe in this note enjoys the same feature. In the sequel we refer to the M_i's that are found in the off-line phase of the attack as "would be forgeries".

1.2 The "real" ISO 9796-1 standard

The actual encoding function that is used in the ISO 9796-1 standard is slightly different than the function μ above. For the same setting of parameters (i.e., modulus of $16z + 1$ bits and $8z$-bit messages), the encoding function – denoted μ_{ISO} – is defined as follows:

$$\begin{array}{rcl} \mu_{\text{ISO}}(m) & = & \bar{s}(m_{\ell-1})\; \tilde{s}(m_{\ell-2})\; m_{\ell-1}\; m_{\ell-2} \\ & & s(m_{\ell-3})\; s(m_{\ell-4})\; m_{\ell-3}\; m_{\ell-4} \end{array}$$

$$\begin{array}{l} \cdots \\ s(m_3)\ s(m_2)\ m_3\ m_2 \\ s(m_1)\ s(m_0)\ m_0\ 6 \end{array}$$

where $\tilde{s}(x)$ denotes the nibble $s(x)$ with the least significant bit flipped (i.e., $\tilde{s}(x) = s(x) \oplus 1$, where $\oplus$ denotes exclusive-or). Namely, for these parameters, the difference between $\mu(m)$ and $\mu_{\text{ISO}}(m)$ is that the lowest bit in the second-most-significant nibble of $\mu_{\text{ISO}}(m)$ is flipped. As before, to sign a message m, one needs to apply the RSA operation to $\mu_{\text{ISO}}(m)$.

One can see that now we cannot simply represent the encoding $\mu_{\text{ISO}}(m)$ as a product $\Gamma \cdot x$ with Γ, x as above. Hence the attack must be modified to apply to this encoding function.

2 Modifying the attack

The modified attack is similar to the one from [1], except that it uses a slightly different structure for Γ and x. In the original attack, the constant Γ consisted of several 1's that were separated by as many 0's as there are bits in x. In the modified attack, we again have a constant Γ which consists of a few 1's separated by many 0's, but this time there are fewer separating 0's.

We start with an example. Consider a 64-bit integer x, which is represented as four 16-bit words $x = abcd$ (so a is the most-significant word of x, b is the second-most-significant, etc.). Also, consider the 144-bit constant $\Gamma = 100010001$, where again each digit represent a 16-bit word. Now consider what happens when we multiply $\Gamma \cdot x$. We have

$$\Gamma \cdot x = \begin{array}{cccccccccc} & & & & & & a & b & c & d \\ & & \cdot & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline & & & & & & a & b & c & d \\ & & & a & b & c & d & & & \\ a & b & c & d & & & & & & \\ \hline a & b & c & e & b & c & e & b & c & d \end{array}$$

where $e = a + d$ (assuming that no carry is generated in the addition $a + d$). Notice that the 16-bit d appears only as the least-significant word of the result, and the 16-bit a appears only as the most-significant word of the result. It is therefore possible to arrange it so that the form of the words a, d be different than the form of the words b, c and e, and this could match the different forms of the least- and most-significant words in the encoded message $\mu_{\text{ISO}}(m)$.

More precisely, we consider three types of 16-bit words. For a 16-bit word x, we say that:

- x is a *valid low word* if it has the form $x = s(u)\ s(v)\ v\ 6$, for some two nibbles u, v.
- x is a *valid middle word* if it has the form $x = s(u)\ s(v)\ u\ v$, for some two nibbles u, v.
- x is a *valid high word* if it has the form $x = \bar{s}(u)\ \tilde{s}(v)\ u\ v$, for some two nibbles u, v.

We note that there are exactly 256 valid low words, 256 valid middle words, and 256 valid high words (since in each case we can arbitrarily choose the nibbles u, v).

In the example above, we needed a to be a valid high word, d to be a valid low word, b and c to be valid middle words, and we also needed $e = a + d$ to be a valid middle word. In Appendix A we list useful combinations of valid words for which the sum is also a valid word. We note the following:

- There are 64 pairs x, y such that x is a valid high word, y is a valid low word, and $z = x + y$ is a valid middle word (this is what we needed for the example above). We call such a pair (x, y) a *high-low pair*.
- There are 84 pairs x, y such that x is a valid high word, y is a valid middle word, and $z = x + y$ is a valid middle word. We call such a pair (x, y) a *high-mid pair*.
- There are 150 pairs x, y such that x is a valid middle word, y is a valid low word, and $z = x + y$ is a valid middle word. We call such a pair (x, y) a *mid-low pair*.
- There are 468 pairs x, y such that x is a valid middle word, y is a valid middle word, and $z = x + y$ is also a valid middle word. We call such a pair (x, y) a *mid-mid pair*.

We are now ready to present the attack. For clarity of presentation we start by presenting the attack for the special cases where the modulus size is 1024+1 bits and 2048+1 bits. We later describe the general case.

2.1 Moduli of size 1024+1 bits

When the modulus size is $k = 1025$ bits, we need to encode the messages as 1024-bit integers with the high bit set to 1. The attack proceeds similarly to the example from above: We consider 64-bit integers $x = abcd$, where a is a valid high-word, d is a valid low-word, and b, c and $e = a + d$ are valid middle words. There are 64 choices for the high-low pair (a, d) and 256 choices for each of b, c, so there are total of 2^{22} x'es of the right form. We then set

$$\Gamma_{1024} \;=\; \sum_{i=0}^{20} 2^{48i} \;=\; \underbrace{1\ 001\ 001\ \ldots\ 001\ {}_{2^{16}}}_{\text{1 followed by 20 repetitions of 001 (base } 2^{16})}$$

This gives us

$$M = \Gamma_{1024} \cdot x \;=\; a\ \underbrace{bce\ bce\ \ldots\ bce}_{\text{20 repetitions}}\ bcd$$

which is a valid encoding of some message $M = \mu_{\text{ISO}}(m)$, because of the way x was chosen. If we set the smoothness bound of the attack to $B = 2^{16}$, then the 64-bit integer x has probability of about $2^{-7.7}$ to be B-smooth, so we expect that there are about $2^{22} \cdot 2^{-7.7} \approx 20000$ x'es of the right form which are B-smooth. Since there are only about 6500 primes smaller than 2^{16}, we have more than enough smooth x'es to get the "homomorphic dependencies" that are needed for the attack.

The above attack has essentially the same complexity as the one that is described by Coron et al. in [1, Section 4.1] (since it uses 64-bit integers x, just as it is done in the original attack). In it reported in [1] that for smoothness bound of about 2^{15}, a single PC can prepare thousands of "would be forgeries" in less than a day. Recall also that this work is all done off-line, and then these "would-be forgeries" can be used against any RSA modulus of 1024+1 bits. After the off-line work is done, the attack needs to collect about 3000 signatures, and then the actual forgeries can be generated instantly.

2.2 Moduli of size 2048+1 bits

When the modulus size is $k = 2049$ bits, we need to encode the messages as 2048-bit integers with the high bit set to 1. Here we need to modify the attack a little, by changing the length of x and the amount of "overlap" that is used in the product $\Gamma \cdot x$. Specifically, we can work with 128-bit x'es, $x = abcdefgh$, where a is a valid high-word, h is a valid low-word, and b, c, d, e, f, g and also $i = a + g$ and $j = b + h$ are valid middle-words. This gives us 84 choices for the high-mid pair (a, g), 150 choices for the mid-low pair (b, h) and 256 choices for each of c, d, e, f, so we have total of more than 2^{45} choices for x. We set

$$\Gamma_{2048} \;=\; \sum_{i=0}^{20} 2^{96i} \;=\; 1\ \underbrace{000001\ \ldots\ 000001}_{20\text{ repetitions}}\ {}_{2^{16}}$$

and so we get

$$M = \Gamma_{2048} \cdot x \;=\; ab\ \underbrace{cdefij\ \ldots\ cdefij}_{20\text{ repetitions}}\ cdefgh$$

which is again a valid encoding. Since x is a 128-bit integer, the probability that it is, say, 2^{20}-smooth is about $2^{-17.4}$. So we expect there to be about $2^{45} \cdot 2^{-17.4} \approx 2^{28}$ x'es of the right form which are 2^{20}-smooth, and we only need about $82000 \approx 2^{16}$ of them to get the "homomorphic dependencies" (since there are about 82000 primes smaller than 2^{20}).

Using the estimates from [1, Section 2], the complexity of finding a B-smooth, L-bit integer x during the off-line phase of this attack is about $C_{L,B} = \mathcal{O}\left(\frac{L\sqrt{B}}{\rho(L/\log_2 B)}\right)$. In our case we have $L = 128, B = 2^{20}$ so we get $C_{L,B} \approx 2^{35}$. In the off-line phase of the attack we need to find about 82000 smooth x'es to get "homomorphic dependencies", and then each additional smooth x would give us another "would be forgery". Hence we estimate that the complexity of the off-line phase is about 2^{51} to get the first "would be forgery", and then 2^{35} for each additional one. This is still well below the complexity of, say, an exhaustive DES key search (and just as for DES key search, this work can be done off-line and is easily parallelizable).

Once the off-line phase is over, the list of "would be forgeries" can be used against any RSA modulus of 2048+1 bits. The attack needs to collect about 82000 signatures, and then the forgeries can be produced almost instantly.

2.3 The general case

For a modulus whose size if $16z+1$ bits (for an even z), we need to encode the messages as $16z$-bit integers, which means that the encodings should have z 16-bit words. We write the integer z as $z = \alpha \cdot m + \beta$, where α, β, m are all integers with $\alpha, \beta \geq 1$ and $m \geq 2$. For reasons that will soon become clear, we try to get $\alpha + \beta$ as small as possible, while making sure that $\alpha - \beta$ is at least 2 or 3.

The attack then works with integers x of $\alpha+\beta$ 16-bit words (which is why we want to minimize $\alpha + \beta$), and use "overlap" of β words in the product $\Gamma \cdot x$. If we denote $\gamma = \alpha + \beta$, then we have $x = a_\gamma \ldots a_1$, where a_γ is a valid high-word, a_1 is a valid low-word, and the other a_i's are valid middle words (and we also need some of the sums to be valid middle words). We then set

$$\Gamma_{16z} \;=\; \sum_{i=0}^{m-1} 2^{16\alpha i} \;=\; 1 \quad \underbrace{0..0\,1 \quad 0..0\,1 \quad \ldots \quad 0..0\,1}_{m-1\text{ repetitions of }0..01\ (\alpha-1\ 0'\text{s followed by }1)}$$

When we multiply $\Gamma_{16z} \cdot x$ we get

$$\Gamma_{16z} \cdot x = \begin{array}{ccccccccccccccccc}
 & & & & & & & a_\gamma & .. & & a_{\alpha+1} & a_\alpha & .. & a_\beta & .. & & a_1 \\
 & & \ldots & 0 & 1 & 0 & & .. & & 0 & 1 & 0 & & .. & & 0 & 1 \\
\hline
 & & & & & & & a_\gamma & .. & & a_{\alpha+1} & a_\alpha & .. & a_\beta & .. & & a_1 \\
 & a_\gamma & .. & & a_{\alpha+1} & a_\alpha & .. & a_\beta & .. & & a_1 & & & & & & \\
\ldots & a_\beta & .. & & a_1 & & & & & & & & & & & & \\
\hline
\end{array}$$

hence we also need the sums $(a_\gamma + a_\beta), \ldots, (a_{\alpha+2} + a_2), (a_{\alpha+1} + a_1)$ to be valid middle words.

If $\beta = 1$ (as in the case of 1025-bit moduli above), we have 64 choices for the high-low pair (a_γ, a_1) and 256 choices for each of the other a_i's, so we get total of $64 \cdot 256^{\alpha-1}$ choices for x.

If $\beta \geq 2$ (as in the case of 2049-bit moduli above), we have 84 choices for the high-mid pair (a_γ, a_β), 150 choices for the mid-low pair $(a_{\alpha+1}, a_1)$, 468 choices for each of the mid-mid pairs $(a_{\gamma-1}, a_{\beta-1}) \ldots (a_{\alpha+2}, a_2)$. Thus the total number of choices for x is $84 \cdot 150 \cdot 468^{\beta-2} \cdot 256^{\alpha-\beta}$. (This is the reason that we want $\alpha - \beta$ to be at least 2 or 3.) For the attack to be successful, we should set the parameters α, β so that there are enough smooth x'es to guarantee the "homomorphic dependencies" that we need.

As another example for the general case, consider moduli of 768+1 bits. We need to encode the messages as integers of 768 bits, or $768/16 = 48$ words. We can write $48 = 5 \cdot 9 + 3$, so we have $\alpha = 5, \beta = 3$. Hence we work with x'es of $5 + 3 = 8$ words (128 bits) and use an overlap of 3 words. For this case we have $84 \cdot 150 \cdot 468 \cdot 256^2 > 2^{38}$ choices for x. If pick the smoothness bound to be 2^{20}, then the probability that x be smooth is about $2^{-17.4}$, so we expect there to be about 2^{21} smooth x'es, and we only need about $82000 \approx 2^{16}$ of them to get the "homomorphic dependencies", since there are about 82000 primes smaller than 2^{20}. The complexity of this attack is the same as for the $(2048 + 1)$-bit moduli.

2.4 Possible extensions

The attack that we described above was intended to works against moduli of size $16z+1$ bits for an even integer z, but there are a few straightforward ways to extend the attack to handle other moduli sizes. For example, for a modulus of size $16z$-bits (with z even), we should encode messages as integers with $16z - 1$ bits, which we can view as z-word integers with the highest bit set to 0 and the second-highest bit set to 1. To handle these integers, we re-define a *valid high-word* as a 16-bit word of the form $x = \hat{s}(u)\ \tilde{s}(v)\ u\ v$, for some two nibbles u, v, where $\hat{s}(u)$ is the nibble $s(u)$ with the highest bit set to 0 and the second-highest bit set to 1. Although we did not check this, we suspect that the modified definition of a valid high-word will not significantly change the number of high-low and high-mid pairs, so the complexity of an attack against $16z$-bit moduli should be roughly the same as that of an attack against moduli of $16z + 1$ bits.

Another extension of the attack is to consider also the cases where there are some carry bits between the nibbles in the computation of $\Gamma \cdot x$. For example, for the case of $\beta \geq 2$ (see Section 2.3) we can have carry bits between the "overlap" words in the multiplication without effecting the attack. We estimate that considering these carry bits can increase the number of possible x'es by about a factor of $2^{\beta-1}$ (since we can have x's that cause any pattern of carry bits inside a string of length β nibbles).

Yet another plausible extension is to handle the case where not only the first and last words of the encoding have different formats, but also one other word in the middle. This is the case, for example, when we encode a message m of length less than half the size of the modulus. In that case, the form of the highest word would be $x = \bar{s}(u)\ s(v)\ u\ v$, the form of the lowest word would be $x = \bar{s}(u)\ s(v)\ v\ 6$, and there would be one other word somewhere in the middle of the form $x = s(u)\ \tilde{s}(v)\ u\ v$. In this case we may be able to modify Γ a little, so that the spacing of the 1's is not equal everywhere. For example, if we have $x = abcd$ and $\Gamma = 10010001$, we get

$$\Gamma \cdot x = \begin{array}{ccccccccccc} & & & & & & & a & b & c & d \\ & & \cdot & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline & & & & & & & a & b & c & d \\ & & & a & b & c & d & & & & \\ a & b & c & d & & & & & & & \\ \hline a & b & c & e & b & c & d & a & b & c & d \end{array}$$

Now notice that the word e only appears once in the middle, and so we can arrange it so that it would have a different form than the other words. This technique can potentially be used to find more forgeries, or to reduce the complexity of the attack against certain moduli-lengths.

3 Conclusions

In this note we demonstrated that the ISO 9796-1 standard can be broken using a variant of the Coron, Naccache and Stern attack from [1]. The estimated complexity of the new attack depends heavily on the modulus length: for some lengths (e.g., 1024+1 bits) the attack can be easily carried out on a single PC in less than a day, while for other lengths (e.g., 2048+1 bits) it has nearly the same complexity as an exhaustive search for a DES key. Still, we stress that the attack is feasible against all the moduli lengths that we considered. We also sketched a few ways in which this attack can be generalized to work against other moduli lengths.

In light of this break, we believe that the standard needs to be modified. In our view, the first step that should be taken is to re-examine the need for this mode of "hash-free encoding" for signatures. An obvious disadvantage of this mode is that it gives an attacker quite a bit of control over the encoded messages. Since the encoding rule is usually very "local" (i.e., each bit of m effects only very few bits of $\mu(m)$), an attacker has an ample opportunity to play with m in order to arrange that $\mu(m)$ has some desired properties. Moreover, as opposed to the "full domain hash" that can be analyzed (and proven secure) in the random-oracle model, there seems to be no hope of getting similar results in the "hash free" case.

If it is decided to keep this "hash free" mode, we describe in Appendix B some possible modifications that can be made to the encoding function. In particular, we suggest to consider encodings with "massive mask changes", such as the functions μ_2 and μ_3 from Subsection B.1. (Similar encodings were also suggested in [8].) These functions stay close to the original intent of ISO 9796-1, but at the same time they seem resistant to multiplicative attacks such as the ones in [1] and in this note.

Acknowledgments. We thank Mike Matyas for encouraging us to work on these new attacks and for several useful discussions. We also thank David Naccache for bringing the note [8] to our attention.

References

[1] Jean-Sébastien Coron, David Naccache and Julien P. Stern. On the Security ofRSA Padding In proceedings of Crypto'99. LNCS vol. 1666, pages 1–18, Springer, 1999.

[2] Y. Desment and A.Odlyzko. A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In Crypto'85, LNCS vol. 218, pp. 516-522. Springer-Verlag, 1986.

[3] L.C. Guillou and J.-J. Quisquater. Precautions Taken Against Various Potential Attacks in ISO/IEC DIS 9796". In EUROCRYPT'90, LNCS vol. 473, Pages 465–473. Springer-Verlag, 1990.

[4] ISO/IEC 9796-1, Information technology – Security techniques – Digital signature schemes giving message recovery – Part 1: Mechanisms using redundancy.

[5] J.F. Misarsky. How (not) to design RSA signature schemes. In PKC'98, LNCS vol. 1431, pp. 14-28. Springer-Verlag, 1998.

[6] M.O. Rabin. Digitized Signatures and public key functions as intractable as factoring. MIT/LCS/TR-212, 1979.

[7] R. Rivest, A. Shamir and L. Adelman. A Method for Obtaining Digital Signature and Public Key Cryptosystems. *Comm. of ACM*, 21 (1978), pp. 120–126

[8] SC27N23xx: Report of the Ad-Hoc meeting on 9796 held at Gemplus, Paris, on May-14-1999.

A Useful pairs for the attack

Here we list all the various types of pairs (x, y) of 16-bit words that we use in our attack (together with their sum, $z = x + y$). All the constants in the tables below are in hexadecimal (base-16) representation.

Table 1: High-Low pairs

$x =$	$8f30$	$af60$	$8f80$	$bfa0$	$afd0$	$b211$	$d221$	9241	$c251$	$d291$	$92f1$	$a462$
$y =$	0316	4316	4316	2266	1316	$0d96$	$1ce6$	$1d96$	$0d96$	$2ce6$	$1ce6$	$3ba6$
$z =$	9246	$f276$	$d296$	$e206$	$c2e6$	$bfa7$	$ef07$	$afd7$	$cfe7$	$ff77$	$afd7$	$e008$
$x =$	$a4d2$	$94f2$	$d923$	9943	8983	$99f3$	8834	$a864$	8884	$b8a4$	$a8d4$	8585
$y =$	$4ba6$	$3ba6$	2456	4456	2456	5316	1316	5316	5316	3266	2316	6086
$z =$	$f078$	$d098$	$fd79$	$dd99$	$add9$	$ed09$	$9b4a$	$fb7a$	$db9a$	$eb0a$	$cbea$	$e60b$
$x =$	$95f5$	$d326$	9346	8386	$93f6$	$ae67$	$aed7$	$9ef7$	8138	8138	9148	$b1a8$
$y =$	6086	2456	4456	2456	5316	$3ba6$	$4ba6$	$3ba6$	$2ba6$	$6ad6$	$3ba6$	$4ad6$
$z =$	$f67b$	$f77c$	$d79c$	$a7dc$	$e70c$	$ea0d$	$fa7d$	$da9d$	$acde$	$ec0e$	$ccee$	$fc7e$
$x =$	$a1d8$	$cc59$	$8c89$	$ba1a$	$8a3a$	$9a4a$	$8a8a$	$caea$	$c75b$	$c7eb$	$97fb$	$b61c$
$y =$	$1ad6$	2526	2526	4456	5456	2456	4456	2316	$1ba6$	$0ba6$	$1ba6$	$1f76$
$z =$	$bcae$	$f17f$	$b1af$	$fe70$	$de90$	$bea0$	$cee0$	$ee00$	$e301$	$d391$	$b3a1$	$d592$
$x =$	$a66c$	$96fc$	$bb1d$	$8b3d$	$9b4d$	$8b8d$	$bbad$	$9bfd$	$cd5e$	$cdee$	$9dfe$	$b01f$
$y =$	$1f76$	$4e06$	$2ce6$	$1d96$	$2d96$	$6ce6$	$1ce6$	$2ce6$	$1ba6$	$0ba6$	$1ba6$	4456
$z =$	$c5e2$	$e502$	$e803$	$a8d3$	$c8e3$	$f873$	$d893$	$c8e3$	$e904$	$d994$	$b9a4$	$f475$
$x =$	$803f$	$904f$	$808f$	$c0ef$								
$y =$	5456	2456	4456	2316								
$z =$	$d495$	$b4a5$	$c4e5$	$e405$								

Table 2: High-Mid pairs

$x=$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$
$y=$	$2e60$	$1ef0$	2361	$13f1$	2562	$15f2$	2863	$18f3$	2964	$19f4$	2465	$14f5$
$z=$	$ee00$	$de90$	$e301$	$d391$	$e502$	$d592$	$e803$	$d893$	$e904$	$d994$	$e405$	$d495$
$x=$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$
$y=$	2266	$12f6$	$2f67$	$1ff7$	2068	$10f8$	$2d69$	$1df9$	$2b6a$	$1bfa$	$266b$	$16fb$
$z=$	$e206$	$d296$	$ef07$	$df97$	$e008$	$d098$	$ed09$	$dd99$	$eb0a$	$db9a$	$e60b$	$d69b$
$x=$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	$bfa0$	9241	9241	9241	9241
$y=$	$276c$	$17fc$	$2a6d$	$1afd$	$2c6e$	$1cfe$	$216f$	$11ff$	4351	2361	$4a5d$	$2a6d$
$z=$	$e70c$	$d79c$	$ea0d$	$da9d$	$ec0e$	$dc9e$	$e10f$	$d19f$	$d592$	$b5a2$	$dc9e$	$bcae$
$x=$	9442	9442	9442	9442	$b9a3$	$b9a3$	$b9a3$	$b9a3$	$b9a3$	$b9a3$	$b9a3$	$b9a3$
$y=$	4552	2562	$465b$	$266b$	$2e60$	$1ef0$	2863	$18f3$	$2d69$	$1df9$	$276c$	$17fc$
$z=$	$d994$	$b9a4$	$da9d$	$baad$	$e803$	$d893$	$e206$	$d296$	$e70c$	$d79c$	$e10f$	$d19f$
$x=$	$b5a5$	$b5a5$	$b5a5$	$b5a5$	$b316$	$b316$	$b316$	$b316$	$c356$	$c356$	$c356$	$c356$
$y=$	$2e60$	$1ef0$	$2b6a$	$1bfa$	$4b5a$	$0b8a$	$415f$	$018f$	$3b1a$	$0b8a$	$311f$	$018f$
$z=$	$e405$	$d495$	$e10f$	$d19f$	$fe70$	$bea0$	$f475$	$b4a5$	$fe70$	$cee0$	$f475$	$c4e5$
$x=$	8386	8386	8386	8386	$b3a6$	$b3a6$	$b3a6$	$b3a6$	$b1a8$	$b1a8$	$b1a8$	$b1a8$
$y=$	$3b1a$	$4b5a$	$311f$	$415f$	$2e60$	$1ef0$	$2d69$	$1df9$	$2e60$	$1ef0$	$2f67$	$1ff7$
$z=$	$bea0$	$cee0$	$b4a5$	$c4e5$	$e206$	$d296$	$e10f$	$d19f$	$e008$	$d098$	$e10f$	$d19f$
$x=$	$b7ab$	$b7ab$	$b7ab$	$b7ab$	$bbad$	$bbad$	$bbad$	$bbad$	$bdae$	$bdae$	$bdae$	$bdae$
$y=$	$2e60$	$1ef0$	2964	$19f4$	$2e60$	$1ef0$	2562	$15f2$	$2e60$	$1ef0$	2361	$13f1$
$z=$	$e60b$	$d69b$	$e10f$	$d19f$	$ea0d$	$da9d$	$e10f$	$d19f$	$ec0e$	$dc9e$	$e10f$	$d19f$

Table 3: Mid-Low pairs

$x =$	5e20	9e40	4e50	2e60	0e80	1ef0	2361	0381	a3d1	13f1	3512	5522
$y =$	9456	3456	6456	3456	9456	c316	cba6	5ba6	5ba6	cba6	2ba6	1ba6
$z =$	f276	d296	b2a6	62b6	a2d6	e206	ef07	5f27	ff77	df97	60b8	70c8
$x =$	8532	8532	9542	4552	2562	0582	b5a2	75c2	a5d2	15f2	3813	5823
$y =$	1ba6	5ad6	2ba6	0ad6	5ad6	3ad6	3ad6	5ad6	0ad6	5ad6	5526	4526
$z =$	a0d8	e008	c0e8	5028	8038	4058	f078	d098	b0a8	70c8	8d39	9d49
$x =$	4853	0883	68b3	68b3	78c3	18f3	3914	3914	8934	9944	4954	4954
$y =$	b526	b526	4526	8456	5526	8456	c266	b1f6	5266	2266	2266	51f6
$z =$	fd79	bda9	add9	ed09	cde9	9d49	fb7a	eb0a	db9a	bbaa	6bba	9b4a
$x =$	2964	2964	0984	0984	b9a4	69b4	19f4	3415	3415	8435	9445	4455
$y =$	5266	21f6	c266	f1f6	21f6	51f6	b1f6	c266	b1f6	5266	2266	2266
$z =$	7bca	4b5a	cbea	fb7a	db9a	bbaa	cbea	f67b	e60b	d69b	b6ab	66bb
$x =$	4455	2465	2465	0485	0485	b4a5	64b5	14f5	3216	5226	4256	0286
$y =$	51f6	5266	21f6	c266	f1f6	21f6	51f6	b1f6	5526	4526	b526	b526
$z =$	964b	76cb	465b	c6eb	f67b	d69b	b6ab	c6eb	873c	974c	f77c	b7ac
$x =$	62b6	62b6	72c6	12f6	3f17	5f27	8f37	8f37	9f47	4f57	2f67	0f87
$y =$	4526	8456	5526	8456	2ba6	1ba6	1ba6	5ad6	2ba6	0ad6	5ad6	3ad6
$z =$	a7dc	e70c	c7ec	974c	6abd	7acd	aadd	ea0d	caed	5a2d	8a3d	4a5d
$x =$	bfa7	7fc7	afd7	1ff7	2068	0088	a0d8	10f8	5d29	9d49	4d59	2d69
$y =$	3ad6	5ad6	0ad6	5ad6	cba6	5ba6	5ba6	cba6	9456	3456	6456	3456
$z =$	fa7d	da9d	baad	7acd	ec0e	5c2e	fc7e	dc9e	f17f	d19f	b1af	61bf
$x =$	0d89	1df9	3b1a	5b2a	4b5a	0b8a	db9a	6bba	7bca	7bca	1bfa	361b
$y =$	9456	c316	5316	4316	b316	b316	1266	4316	5316	1266	1266	2d96
$z =$	a1df	e10f	8e30	9e40	fe70	bea0	ee00	aed0	cee0	8e30	2e60	63b1
$x =$	361b	562b	863b	964b	465b	266b	068b	b6ab	66bb	16fb	371c	572c
$y =$	ace6	1d96	1d96	2d96	4ce6	1ce6	ece6	1ce6	4ce6	ace6	1e06	2e06
$z =$	e301	73c1	a3d1	c3e1	9341	4351	f371	d391	b3a1	c3e1	5522	8532
$x =$	873c	276c	276c	078c	078c	67bc	77cc	a7dc	a7dc	17fc	3a1d	5a2d
$y =$	0e06	ce06	bd96	ce06	4d96	0e06	2e06	1e06	4d96	bd96	1e06	2e06
$z =$	9542	f572	e502	d592	5522	75c2	a5d2	c5e2	f572	d592	5823	8833
$x =$	8a3d	2a6d	2a6d	0a8d	0a8d	6abd	7acd	aadd	aadd	1afd	3c1e	3c1e
$y =$	0e06	ce06	bd96	ce06	4d96	0e06	2e06	1e06	4d96	bd96	2d96	ace6
$z =$	9843	f873	e803	d893	5823	78c3	a8d3	c8e3	f873	d893	69b4	e904
$x =$	5c2e	8c3e	9c4e	4c5e	2c6e	0c8e	bcae	6cbe	1cfe	311f	512f	415f
$y =$	1d96	1d96	2d96	4ce6	1ce6	ece6	1ce6	4ce6	ace6	5316	4316	b316
$z =$	79c4	a9d4	c9e4	9944	4954	f974	d994	b9a4	c9e4	8435	9445	f475
$x =$	018f	d19f	61bf	71cf	71cf	11ff						
$y =$	b316	1266	4316	5316	1266	1266						
$z =$	b4a5	e405	a4d5	c4e5	8435	2465						

Table 4: Mid-Mid pairs (Part 1)

$x =$	3311	3311	3311	3311	5321	5321	5321	5321	9341	9341	9341	9341
$y =$	5522	75c2	572c	77cc	3512	75c2	371c	77cc	4552	2562	475c	276c
$z =$	8833	a8d3	8a3d	aadd	8833	c8e3	8a3d	caed	d893	b8a3	da9d	baad
$x =$	4351	4351	4351	4351	2361	2361	2361	2361	73c1	73c1	73c1	73c1
$y =$	9542	2562	974c	276c	9542	4552	974c	475c	3512	5522	371c	572c
$z =$	d893	68b3	da9d	6abd	b8a3	68b3	baad	6abd	a8d3	c8e3	aadd	caed
$x =$	3512	3512	3512	3512	5522	5522	5522	5522	9542	9542	9542	9542
$y =$	5321	73c1	572c	77cc	3311	73c1	371c	77cc	4351	2361	475c	276c
$z =$	8833	a8d3	8c3e	acde	8833	c8e3	8c3e	ccee	d893	b8a3	dc9e	bcae
$x =$	4552	4552	4552	4552	2562	2562	2562	2562	75c2	75c2	75c2	75c2
$y =$	9341	2361	974c	276c	9341	4351	974c	475c	3311	5321	371c	572c
$z =$	d893	68b3	dc9e	6cbe	b8a3	68b3	bcae	6cbe	a8d3	c8e3	acde	ccee
$x =$	3914	3914	3914	3914	3914	3914	3914	3914	5924	5924	5924	5924
$y =$	5425	74c5	5226	72c6	5a2d	6abd	5c2e	6cbe	3415	74c5	3216	72c6
$z =$	8d39	add9	8b3a	abda	9341	a3d1	9542	a5d2	8d39	cde9	8b3a	cbea
$x =$	5924	5924	5924	5924	5924	5924	5924	5924	8934	8934	8934	8934
$y =$	3a1d	9a4d	0a8d	6abd	3c1e	9c4e	0c8e	6cbe	4a5d	2a6d	4c5e	2c6e
$z =$	9341	f371	63b1	c3e1	9542	f572	65b2	c5e2	d391	b3a1	d592	b5a2
$x =$	9944	9944	9944	9944	9944	9944	9944	9944	4954	4954	4954	4954
$y =$	4455	2465	4256	2266	5a2d	0a8d	5c2e	0c8e	9445	2465	9246	2266
$z =$	dd99	bda9	db9a	bbaa	f371	a3d1	f572	a5d2	dd99	6db9	db9a	6bba
$x =$	4954	4954	4954	4954	2964	2964	2964	2964	2964	2964	2964	2964
$y =$	8a3d	2a6d	8c3e	2c6e	9445	4455	9246	4256	8a3d	4a5d	8c3e	4c5e
$z =$	d391	73c1	d592	75c2	bda9	6db9	bbaa	6bba	b3a1	73c1	b5a2	75c2
$x =$	0984	0984	0984	0984	69b4	69b4	69b4	69b4	79c4	79c4	79c4	79c4
$y =$	5a2d	9a4d	5c2e	9c4e	3a1d	5a2d	3c1e	5c2e	3415	5425	3216	5226
$z =$	63b1	a3d1	65b2	a5d2	a3d1	c3e1	a5d2	c5e2	add9	cde9	abda	cbea
$x =$	3415	3415	3415	3415	5425	5425	5425	5425	9445	9445	9445	9445
$y =$	5924	79c4	5226	72c6	3914	79c4	3216	72c6	4954	2964	4256	2266
$z =$	8d39	add9	863b	a6db	8d39	cde9	863b	c6eb	dd99	bda9	d69b	b6ab
$x =$	4455	4455	4455	4455	2465	2465	2465	2465	74c5	74c5	74c5	74c5
$y =$	9944	2964	9246	2266	9944	4954	9246	4256	3914	5924	3216	5226
$z =$	dd99	6db9	d69b	66bb	bda9	6db9	b6ab	66bb	add9	cde9	a6db	c6eb
$x =$	3216	3216	3216	3216	5226	5226	5226	5226	9246	9246	9246	9246
$y =$	5924	79c4	5425	74c5	3914	79c4	3415	74c5	4954	2964	4455	2465
$z =$	8b3a	abda	863b	a6db	8b3a	cbea	863b	c6eb	db9a	bbaa	d69b	b6ab
$x =$	4256	4256	4256	4256	2266	2266	2266	2266	72c6	72c6	72c6	72c6
$y =$	9944	2964	9445	2465	9944	4954	9445	4455	3914	5924	3415	5425
$z =$	db9a	6bba	d69b	66bb	bbaa	6bba	b6ab	66bb	abda	cbea	a6db	c6eb

Table 4: Mid-Mid pairs (Part 2)

$x =$	3f17	3f17	3f17	3f17	5f27	5f27	5f27	5f27	5f27	5f27	5f27	5f27
$y =$	562b	66bb	5a2d	6abd	361b	964b	068b	66bb	3a1d	9a4d	0a8d	6abd
$z =$	9542	a5d2	9944	a9d4	9542	f572	65b2	c5e2	9944	f974	69b4	c9e4
$x =$	8f37	8f37	8f37	8f37	9f47	9f47	9f47	9f47	4f57	4f57	4f57	4f57
$y =$	465b	266b	4a5d	2a6d	562b	068b	5a2d	0a8d	863b	266b	8a3d	2a6d
$z =$	d592	b5a2	d994	b9a4	f572	a5d2	f974	a9d4	d592	75c2	d994	79c4
$x =$	2f67	2f67	2f67	2f67	0f87	0f87	0f87	0f87	6fb7	6fb7	6fb7	6fb7
$y =$	863b	465b	8a3d	4a5d	562b	964b	5a2d	9a4d	361b	562b	3a1d	5a2d
$z =$	b5a2	75c2	b9a4	79c4	65b2	a5d2	69b4	a9d4	a5d2	c5e2	a9d4	c9e4
$x =$	3d19	3d19	3d19	3d19	5d29	5d29	5d29	5d29	5d29	5d29	5d29	5d29
$y =$	5b2a	6bba	572c	67bc	3b1a	9b4a	0b8a	6bba	371c	974c	078c	67bc
$z =$	9843	a8d3	9445	a4d5	9843	f873	68b3	c8e3	9445	f475	64b5	c4e5
$x =$	8d39	8d39	8d39	8d39	9d49	9d49	9d49	9d49	4d59	4d59	4d59	4d59
$y =$	4b5a	2b6a	475c	276c	5b2a	0b8a	572c	078c	8b3a	2b6a	873c	276c
$z =$	d893	b8a3	d495	b4a5	f873	a8d3	f475	a4d5	d893	78c3	d495	74c5
$x =$	2d69	2d69	2d69	2d69	0d89	0d89	0d89	0d89	6db9	6db9	6db9	6db9
$y =$	8b3a	4b5a	873c	475c	5b2a	9b4a	572c	974c	3b1a	5b2a	371c	572c
$z =$	b8a3	78c3	b4a5	74c5	68b3	a8d3	64b5	a4d5	a8d3	c8e3	a4d5	c4e5
$x =$	3b1a	3b1a	3b1a	3b1a	5b2a	5b2a	5b2a	5b2a	5b2a	5b2a	5b2a	5b2a
$y =$	5d29	6db9	572c	67bc	3d19	9d49	0d89	6db9	371c	974c	078c	67bc
$z =$	9843	a8d3	9246	a2d6	9843	f873	68b3	c8e3	9246	f276	62b6	c2e6
$x =$	8b3a	8b3a	8b3a	8b3a	9b4a	9b4a	9b4a	9b4a	4b5a	4b5a	4b5a	4b5a
$y =$	4d59	2d69	475c	276c	5d29	0d89	572c	078c	8d39	2d69	873c	276c
$z =$	d893	b8a3	d296	b2a6	f873	a8d3	f276	a2d6	d893	78c3	d296	72c6
$x =$	2b6a	2b6a	2b6a	2b6a	0b8a	0b8a	0b8a	0b8a	6bba	6bba	6bba	6bba
$y =$	8d39	4d59	873c	475c	5d29	9d49	572c	974c	3d19	5d29	371c	572c
$z =$	b8a3	78c3	b2a6	72c6	68b3	a8d3	62b6	a2d6	a8d3	c8e3	a2d6	c2e6
$x =$	361b	361b	361b	361b	562b	562b	562b	562b	562b	562b	562b	562b
$y =$	5f27	6fb7	5a2d	6abd	3f17	9f47	0f87	6fb7	3a1d	9a4d	0a8d	6abd
$z =$	9542	a5d2	9048	a0d8	9542	f572	65b2	c5e2	9048	f078	60b8	c0e8
$x =$	863b	863b	863b	863b	964b	964b	964b	964b	465b	465b	465b	465b
$y =$	4f57	2f67	4a5d	2a6d	5f27	0f87	5a2d	0a8d	8f37	2f67	8a3d	2a6d
$z =$	d592	b5a2	d098	b0a8	f572	a5d2	f078	a0d8	d592	75c2	d098	70c8
$x =$	266b	266b	266b	266b	068b	068b	068b	068b	66bb	66bb	66bb	66bb
$y =$	8f37	4f57	8a3d	4a5d	5f27	9f47	5a2d	9a4d	3f17	5f27	3a1d	5a2d
$z =$	b5a2	75c2	b0a8	70c8	65b2	a5d2	60b8	a0d8	a5d2	c5e2	a0d8	c0e8
$x =$	371c	371c	371c	371c	371c	371c	371c	371c	572c	572c	572c	572c
$y =$	5321	73c1	5522	75c2	5d29	6db9	5b2a	6bba	3311	73c1	3512	75c2
$z =$	8a3d	aadd	8c3e	acde	9445	a4d5	9246	a2d6	8a3d	caed	8c3e	ccee

Table 4: Mid-Mid pairs (Part 3)

$x =$	$572c$	$572c$	$572c$	$572c$	$572c$	$572c$	$572c$	$572c$	$873c$	$873c$	$873c$	$873c$
$y =$	$3d19$	$9d49$	$0d89$	$6db9$	$3b1a$	$9b4a$	$0b8a$	$6bba$	$4d59$	$2d69$	$4b5a$	$2b6a$
$z =$	9445	$f475$	$64b5$	$c4e5$	9246	$f276$	$62b6$	$c2e6$	$d495$	$b4a5$	$d296$	$b2a6$
$x =$	$974c$	$974c$	$974c$	$974c$	$974c$	$974c$	$974c$	$974c$	$475c$	$475c$	$475c$	$475c$
$y =$	4351	2361	4552	2562	$5d29$	$0d89$	$5b2a$	$0b8a$	9341	2361	9542	2562
$z =$	$da9d$	$baad$	$dc9e$	$bcae$	$f475$	$a4d5$	$f276$	$a2d6$	$da9d$	$6abd$	$dc9e$	$6cbe$
$x =$	$475c$	$475c$	$475c$	$475c$	$276c$	$276c$	$276c$	$276c$	$276c$	$276c$	$276c$	$276c$
$y =$	$8d39$	$2d69$	$8b3a$	$2b6a$	9341	4351	9542	4552	$8d39$	$4d59$	$8b3a$	$4b5a$
$z =$	$d495$	$74c5$	$d296$	$72c6$	$baad$	$6abd$	$bcae$	$6cbe$	$b4a5$	$74c5$	$b2a6$	$72c6$
$x =$	$078c$	$078c$	$078c$	$078c$	$67bc$	$67bc$	$67bc$	$67bc$	$77cc$	$77cc$	$77cc$	$77cc$
$y =$	$5d29$	$9d49$	$5b2a$	$9b4a$	$3d19$	$5d29$	$3b1a$	$5b2a$	3311	5321	3512	5522
$z =$	$64b5$	$a4d5$	$62b6$	$a2d6$	$a4d5$	$c4e5$	$a2d6$	$c2e6$	$aadd$	$caed$	$acde$	$ccee$
$x =$	$3a1d$	$3a1d$	$3a1d$	$3a1d$	$3a1d$	$3a1d$	$3a1d$	$3a1d$	$5a2d$	$5a2d$	$5a2d$	$5a2d$
$y =$	5924	$69b4$	$5f27$	$6fb7$	$562b$	$66bb$	$5c2e$	$6cbe$	3914	9944	0984	$69b4$
$z =$	9341	$a3d1$	9944	$a9d4$	9048	$a0d8$	$964b$	$a6db$	9341	$f371$	$63b1$	$c3e1$
$x =$	$5a2d$	$5a2d$	$5a2d$	$5a2d$	$5a2d$	$5a2d$	$5a2d$	$5a2d$	$5a2d$	$5a2d$	$5a2d$	$5a2d$
$y =$	$3f17$	$9f47$	$0f87$	$6fb7$	$361b$	$964b$	$068b$	$66bb$	$3c1e$	$9c4e$	$0c8e$	$6cbe$
$z =$	9944	$f974$	$69b4$	$c9e4$	9048	$f078$	$60b8$	$c0e8$	$964b$	$f67b$	$66bb$	$c6eb$
$x =$	$8a3d$	$8a3d$	$8a3d$	$8a3d$	$8a3d$	$8a3d$	$8a3d$	$8a3d$	$9a4d$	$9a4d$	$9a4d$	$9a4d$
$y =$	4954	2964	$4f57$	$2f67$	$465b$	$266b$	$4c5e$	$2c6e$	5924	0984	$5f27$	$0f87$
$z =$	$d391$	$b3a1$	$d994$	$b9a4$	$d098$	$b0a8$	$d69b$	$b6ab$	$f371$	$a3d1$	$f974$	$a9d4$
$x =$	$9a4d$	$9a4d$	$9a4d$	$9a4d$	$4a5d$	$4a5d$	$4a5d$	$4a5d$	$4a5d$	$4a5d$	$4a5d$	$4a5d$
$y =$	$562b$	$068b$	$5c2e$	$0c8e$	8934	2964	$8f37$	$2f67$	$863b$	$266b$	$8c3e$	$2c6e$
$z =$	$f078$	$a0d8$	$f67b$	$a6db$	$d391$	$73c1$	$d994$	$79c4$	$d098$	$70c8$	$d69b$	$76cb$
$x =$	$2a6d$	$2a6d$	$2a6d$	$2a6d$	$2a6d$	$2a6d$	$2a6d$	$2a6d$	$0a8d$	$0a8d$	$0a8d$	$0a8d$
$y =$	8934	4954	$8f37$	$4f57$	$863b$	$465b$	$8c3e$	$4c5e$	5924	9944	$5f27$	$9f47$
$z =$	$b3a1$	$73c1$	$b9a4$	$79c4$	$b0a8$	$70c8$	$b6ab$	$76cb$	$63b1$	$a3d1$	$69b4$	$a9d4$
$x =$	$0a8d$	$0a8d$	$0a8d$	$0a8d$	$6abd$	$6abd$	$6abd$	$6abd$	$6abd$	$6abd$	$6abd$	$6abd$
$y =$	$562b$	$964b$	$5c2e$	$9c4e$	3914	5924	$3f17$	$5f27$	$361b$	$562b$	$3c1e$	$5c2e$
$z =$	$60b8$	$a0d8$	$66bb$	$a6db$	$a3d1$	$c3e1$	$a9d4$	$c9e4$	$a0d8$	$c0e8$	$a6db$	$c6eb$
$x =$	$3c1e$	$3c1e$	$3c1e$	$3c1e$	$5c2e$	$5c2e$	$5c2e$	$5c2e$	$5c2e$	$5c2e$	$5c2e$	$5c2e$
$y =$	5924	$69b4$	$5a2d$	$6abd$	3914	9944	0984	$69b4$	$3a1d$	$9a4d$	$0a8d$	$6abd$
$z =$	9542	$a5d2$	$964b$	$a6db$	9542	$f572$	$65b2$	$c5e2$	$964b$	$f67b$	$66bb$	$c6eb$
$x =$	$8c3e$	$8c3e$	$8c3e$	$8c3e$	$9c4e$	$9c4e$	$9c4e$	$9c4e$	$4c5e$	$4c5e$	$4c5e$	$4c5e$
$y =$	4954	2964	$4a5d$	$2a6d$	5924	0984	$5a2d$	$0a8d$	8934	2964	$8a3d$	$2a6d$
$z =$	$d592$	$b5a2$	$d69b$	$b6ab$	$f572$	$a5d2$	$f67b$	$a6db$	$d592$	$75c2$	$d69b$	$76cb$
$x =$	$2c6e$	$2c6e$	$2c6e$	$2c6e$	$0c8e$	$0c8e$	$0c8e$	$0c8e$	$6cbe$	$6cbe$	$6cbe$	$6cbe$
$y =$	8934	4954	$8a3d$	$4a5d$	5924	9944	$5a2d$	$9a4d$	3914	5924	$3a1d$	$5a2d$
$z =$	$b5a2$	$75c2$	$b6ab$	$76cb$	$65b2$	$a5d2$	$66bb$	$a6db$	$a5d2$	$c5e2$	$a6db$	$c6eb$

B Possible countermeasures

Below we examine a few possible modifications that can be made to the encoding function to protect it against attacks such the ones described in this note.

B.1 Massive mask changes

Supposed that instead of changing one bit here and there in the encoded message, the standard is rewritten to have a massive, fixed, change in $\mu(m)$. For example, let π_i, e_i be the i'th nibbles in the hexadecimal expansion of the irrational numbers $\pi = 3.14159\ldots$ and $e = 2.71828\ldots$, respectively. Possible encodings that use these masks could be:

$$
\begin{aligned}
\mu_1(m) \quad = \quad & \pi_{\ell-1}\ \pi_{\ell-2}\ m_{\ell-1}\ m_{\ell-2} \\
& \pi_{\ell-3}\ \pi_{\ell-4}\ m_{\ell-3}\ m_{\ell-4} \\
& \ldots \\
& \pi_1\ \pi_0\ m_1\ m_0
\end{aligned}
$$

$$
\begin{aligned}
\mu_2(m) \quad = \quad & (\pi_{\ell-1} \oplus s(m_{\ell-1}))\ (\pi_{\ell-2} \oplus s(m_{\ell-2}))\ m_{\ell-1}\ m_{\ell-2} \\
& (\pi_{\ell-3} \oplus s(m_{\ell-3}))\ (\pi_{\ell-4} \oplus s(m_{\ell-4}))\ m_{\ell-1}\ m_{\ell-2} \\
& \ldots \\
& (\pi_1 \oplus s(m_1))\ (\pi_0 \oplus s(m_0))\ m_1\ m_0
\end{aligned}
$$

$$
\begin{aligned}
\mu_3(m) \quad = \quad & (\pi_{\ell-1} \oplus s(m_{\ell-1} \oplus e_{\ell-1}))\ (\pi_{\ell-2} \oplus s(m_{\ell-2} \oplus e_{\ell-2}))\ m_{\ell-1}\ m_{\ell-2} \\
& (\pi_{\ell-3} \oplus s(m_{\ell-3} \oplus e_{\ell-3}))\ (\pi_{\ell-4} \oplus s(m_{\ell-4} \oplus e_{\ell-4}))\ m_{\ell-3}\ m_{\ell-4} \\
& \ldots \\
& (\pi_1 \oplus s(m_1 \oplus e_1))\ (\pi_0 \oplus s(m_0 \oplus e_0))\ m_1\ m_0
\end{aligned}
$$

For each of these, it seems much harder to find a systematic choice of Γ, x and messages m to satisfy $\mu(m) = \Gamma \cdot x$. Below we describe some potential "partial attacks" against these encodings.

It is conceivable that an attacker can find large Γ (say Γ larger than the $3/4$ power of the modulus N), for which there are two strings x and x' and two messages m and m' such that

$$\mu(m) = \Gamma \cdot x \ \text{ and } \ \mu(m') = \Gamma \cdot x'$$

Here Γ would be unstructured (unlike in the attacks from above). Since x and x' would both be relatively small (of size N/Γ), they might both be smooth. Then one could combine the two signatures to eliminate the factor of Γ and develop a relation among the small primes represented by x and x'. We note that for the purpose of this attack, a multiplier Γ is only useful if we can find at least two strings x, x' which satisfy relations as above. Heuristically, we can assert that many such Γ factors EXIST, even very large ones (say, larger than the 9/10 power of N). However, finding them might be quite difficult. We could not come up with any efficient way of finding a suitable Γ and the associated x, x', m, m'.

One possibility that we looked at, is to take two messages m, m' that have small Hamming distance from each other, consider the difference $\Delta = \mu(m) - \mu(m')$ (which also has a low Hamming weight, since the encoding μ is very local), and find an integer Γ which is a factor of the difference Δ. Then alter the nibbles in which m and m' AGREE, creating new messages M, M', in the hope that $\mu(M)$ is a multiple of Γ. If so, then $\mu(M')$ must also be a multiple of Γ. However, there seems to be no efficient way of changing m to M as above; having found Γ by this fairly random process, there is a very slim hope that any M will exist for which $\mu(M)$ is divisible by Γ. So this strategy seem to require quite a bit of trial and error.

(We note that one might hope to make a heuristic argument that encodings as above might be difficult to attack; that if we could find many smooth numbers x satisfying these relations, then we could just as easily factor N in the first place. We have not looked into substantiating such arguments.)

B.2 Length expanding encoding

Other constructions that may be considered, involve encoding the message m into a string longer than the modulus N. This has the advantage that it forces the attacker to deal with larger integers (and so it is potentially harder to find smooth integers), but it does not have the "message recovery" property. That is, it is no longer possible to extract m from the signature on m. For example, suppose that we fix two constants c_0 and c_1, each half the length of the modulus N. To encode a message m (which is also half the length of the modulus N), we form the sums $m + c_0, m + c_1$, express them as binary strings, and form the concatenated string

$$\mu_4(m) = (m + c_0)\,(m + c_1)\,m$$

whose length is 3/2 that of N. For this encoding, an attacker could try to get either $\mu_4(m)$ or the residue $(\mu_4(m) \bmod N)$ to be smooth. Trying to get $\mu_4(m)$ to be smooth, one could set

$$\begin{aligned}
&\alpha = c_0 \cdot 2^{2n} + c_1 \cdot 2^n \\
&\beta = 2^{2n} + 2^n + 1 \\
&\text{and then } \mu_4(m) = \alpha + \beta \cdot m
\end{aligned}$$

As long as α and β are relatively primes, it would seem unlikely that m could be chosen much smaller than $\sqrt{N}$ to obtain $\mu_4(m) = \Gamma \cdot x$ for a large Γ and a small, smooth, x. (Intuitively, it seems hard to pick a small m so that $\mu_4(m)$ has a large factor which we understand and a small factor that we don't, where the small factor is required to be smooth, and that the small factor is smaller than, say, $\sqrt{N}$.)

If the attacker is trying to get the residue $\mu_4(m) \bmod N$ to be smooth, than he can compute a constant $\delta = \alpha/\beta \mod N$, and he is reduced to trying to make the quantity $\delta + m$ smooth. Again this should be a daunting task, since δ is about the same size as N, and m is only about half that size. Hence the attacker can control the low order bits by choice of m, but the uncontrolled part is still of size about $\sqrt{N}$.

A (very informal) hope here is that if the attacker is forced into a position where he needs certain "randomly generated integers" of size $\sqrt{N}$ to be smooth, then he is in no better position than a person applying the Continued Fraction factoring method on N. (Such a person is also generating integers of size $\sqrt{N}$ and depending on a number of them to be smooth for the success of his method.) By contrast, in the attacks from above the attacker works with integers x of 64-128 bits, so the probability that they are smooth is much higher.

B.3 Encoding via squaring

Another encoding method uses addition and squaring: We fix a random constant δ of about the same size as the modulus N, and set

$$\mu_5(m) = m^2 + \delta$$

One advantage of this form is that finding many numbers of the form $\Gamma \cdot x$ for a smooth x, is as difficult as factoring δ (using the "Quadratic Sieve" method). On the other hand, $\mu_5(m)$ is harder to compute than the other encodings, and the relation to the signed message m is not transparent.