Applied Mathematics for Reversers I

An introduction into modular arithmetics and where to use it.
by Haldir[RET]

1.) What is modulo

Most readers already know modulo from Cryptography.
It's used in several algorithms like RSA. Usually % is used in programming languages for modulo.

1.1) A short mathematical introduction to it, using the Integer numbers.

Note: m >= 2, otherwise it's nonsense (m=1)
Binary is a special case (m=2)

The result of a modulo operation is the remainder of a division a/m
-> a mod m = r

a = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
m = 7
r = 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1

Example: 6/7 = 0 R 6, 7/7 = 1 R 0.

Ain't hard, is it ?

Note: in the following section the == means it's a modular congruency relation, in Math books it's often
refered as a = with 3 - instead of two in the normal =

a == b (mod m)

A example could be m = 7, a = 8 , b = 15
In our example we see both values have a remainder of 1.

Multiplication and Addition can be also used as an operator for modulo:
a == b (mod m) AND c == d (mod m)

-> a + c == b + d (mod m) AND a * c == b * d (mod m)

Let's say: The Greatest Common Divisor GCD(c,m) = 1 (means their biggest common divisor is 1)
GCD of two primes is always 1.

a*c == b*c (mod m) -> a = b (mod m)

NOTE: This just applies if GCD(c,m) = 1

What does that all mean ?
It doesn't matter if you do ((a (mod m)) + (b (mod m))) (mod m) or a+b (mod m)

That should be enough for our short math introduction.

2.) An Example where we might use it, compared to non-modulo

Sometimes you might see something like this:

```
uchar m_1[] = { 0x04,0x00,0x0B,0x0E,0x08 };
uchar m_2[] = { 0x07,0x05,0x0C,0x02,0x0A };
uchar m_3[] = { 0x09,0x0E,0x00,0x05,0x08 };
uchar m_4[] = { 0x0A,0x01,0x0E,0x03,0x02 };
uchar m_5[] = { 0x0C,0x0D,0x00,0x0A,0x05 };

uchar *multipliers[] = { m1_1, m1_2, m1_3, m1_4, m1_5 };

for (i=0; i<sizeof(multipliers)/4; i++)
{
multiplier = multipliers[i];
for (j=0,sum=0; j<sizeof(m_1); j++)
sum += (ushort)(mod1[j] * multiplier[j]);
mod2[i] = sum;
}
```

What we have is the Array mod2.
What we want is the Array mod1.

Most should have realized by now that this is a System of Linear Equations
5 Linear Equations for 5 Variables can be solved unique (Only one Result)

How you do that, should be known, if not a short idea:

those m_x[] Arrays can be displayed as a 5*5 Matrix (A) and our mod2 Array is a Vector of Length 5 (x)
our wanted Array mod1 can be also displayed as Vector of Length 5 (b)

Now: A*b = x
You can solve that using the known Gaussian Elimination algorithm or several other methods (i won't discuss these here)

Now, what happens if our Array mod2 has just 3 entries.
Now we can't solve this System of Linear Equations to one unique solution. Simply because we don't have enough
"Information" to reconstruct the missing two Array entries.
If your Matrix and Vectors are small (Length/Dimension is short) you can solve these Equations manually and you need
to bruteforce the "unknown" Values, if it's not small, even the output of some Math Program, like Maple,Mathlab etc.
won't help you that much. We're stuck here.
We also just use 3 columns and 3 rows of our previous 5*5 Matrix, so we really lose "Information" here. We lose
2 rows of Information and we can't do anything about it.

Now what happens if the whole System is "modulo" (means using modular Arithmetics)

```
for (i=0; i<sizeof(multipliers)/4; i++)
{
multiplier = multipliers[i];
for (j=0,sum=0; j<sizeof(m_1); j++)
sum += (ushort)(mod1[j] * multiplier[j]);
mod2[i] = sum % 0x0F; // Here you see our Modulo Operator %
}
```

Well if the Matrix is again 5*5, both Vectors are 5 long, we can solve that like our first example
(No differences, except you need to do mod m each step)

Probably you're wondering why the hell i'm writing about it again:

If we have our second problem and the Result Vector is not of Length 5 but Length 3 (again the missing 2 values)
We again have our 3*3 Matrix, we can use, but we can now resize it to a 5*5 Matrix. HOW ?

We just use our old 5*5 Matrix (or we can alternatively, for easier calculations, replace the last two rows with 0
and just place a 1 in the appropriate diagonal value A[i][i], because for the Gaussian Elimination Algo you need to have
a lower triangular matrix (row echelon form) (that should ring a bell in your head if you know the Gaussian Elimination)

And we add to our 3 element Vector two more values to a Vector of Length 5.

Now you probably ask why ?

Well easy for each missing value in the array (in our example we have 2) there are just m possible Values (because it's all modulo)
so it doesn't matter if you do 500*2000 (mod m) or just 5*20 (mod m), it's the same as if you would do 10 (mod m).
So what we do is, we again bruteforce the values in our A*b = x system, we just try every possible value between 0 and m-1

Now we find a solution in max. m^n steps. m being the modulo value and n is the amount of "missing" vector elements.
In our example it would be F^2 = 15*15 = 255.

Some application notes: This Matrix multiplication in a modulo environment is often used in "cryptographic" algorithms.
You can easily see if these algos are "secure" or not. Just check if the reversed algo contains "jumps" in Vector Lengths (like our Vector
with Length 3 and Vector with Length 5 and the missing "Information" about it.) Now if you have Vectors with many missing elements
and a big modulo (bignum modulo or similar) it would be again something like m=1234567898654321 and n = 1000 (very bad case)
we would have about m^n possible combinations, which is above everything we can ever solve. Then there are still some possible ways
to solve these Problems, but i won't cover them here as this is way too advanced mathematics.

3.) Math Libraries

Most of you already know Libraries like MiRACL or crypto++, they have the basic functions for doing crypto mathematics, but what if you need

some more advanced Library for Math Stuff, like our generic modular Arithmetics. Well you could either use Maple,Matlab etc. or use some ready made Library for that, there are several out there, like Lidia or GMP, but most are just for Linux or won't compile correctly with MSVC.
I would recommend NTL (Number Theory Library) by Victor Shoup (http://shoup.net/ntl/) for solving modular Arithmetics.
It's very fast and easy to use and compiles without problems on about every C++ Compiler (including MSVC). One thing you need to know is that NTL does not do A*b = x for Linear Equation Solving, but b*A = x, this is not the same, so you need to transpose() the Matrix before using the solve() function.

4.) Resumée

As I showed in this essay, modular Arithmetics is not that hard at all and might even help us solving some otherwise "impossible" problems.
I hope you'll understand now the basics about it and might even remember that "two for-loops" might just be a Linear Equation System, if you see something like this in your next Reversing Project. I might write more essays about "applied maths for reversers", which probably won't discuss the basics of cryptography, because you all should read Applied Cryptography :), but I'll focus more on some special cases, which might be interesting.

Haldir[RET]
12/10/2002