Problem Set 3 Solutions

Problem 3-1. All your PRIMES is belong to us

This solution comes from Kenny C.K. Fong, Jonathan M. Hunt, Soyini D. Liburd, and Eduardo I. McLean. Their solutions have been slightly edited.

(a) Generators of Z_p^*

See `http://web.mit.edu/6.857/www/handouts/H18/` for the source code from the TAs and the 6.857 student group.

(b) Expected amount of work to find primes and generators

We accepted a wide range of solutions as long as they had non-trivial lower and upper bounds. We had hoped that you would produce a lower bound based on the $\phi(n) \geq \frac{n}{6 \ln \ln n}$ formula. Most people recognized this. The upper bound, however, is more interesting if we only select co-Sophie-Germain primes.

Some groups did not realize that to efficiently determine the order of an element (and hence, whether it's a generator), we need the the prime factorization of $p-1$. No one yet knows of an efficient algorithm to do this without the prime factorization of $p-1$.

Here is the solution from the student group:

Let p be a prime. Then $\phi(p) = p - 1$, where ϕ is Euler's phi function. We know from part (a) that the number of generators of $\mathbb{Z}_p^*$ is $\phi(\phi(p)) = \phi(p-1)$. Moreover, $|\mathbb{Z}_p^*| = p - 1$. Therefore, for a random prime p, the probability of a random element $g \in \mathbb{Z}_p$ being a generator of $\mathbb{Z}_p^*$ is $\frac{\phi(p-1)}{p-1}$.

From class we know the theorem that $\phi(n) > \frac{n}{6 \ln \ln n}$ for all positive integers n. Furthermore, since p is b-bit long, we have that $\lg(p-1) = b$. We then obtain

$$\phi(p-1) > \frac{p-1}{6 \ln \ln(p-1)} = \frac{p-1}{6 \ln \frac{\lg(p-1)}{\lg e}} = \frac{p-1}{6 \ln \frac{b}{\lg e}} > \frac{p-1}{6 \ln b}.$$

To obtain an upper bound for $\phi(p-1)$, notice that $p-1$ is even. Thus $\phi(p-1) = \phi(2 \cdot \frac{p-1}{2}) = \phi(2)\phi(\frac{p-1}{2}) = \phi(\frac{p-1}{2})$ as $\phi(2) = 1$. Now, notice that $\phi(\frac{p-1}{2})$ is maximized when $\frac{p-1}{2}$ is a prime (i.e. p is a safe prime), in which case $\phi(\frac{p-1}{2}) = \frac{p-1}{2} - 1 = \frac{p-3}{2}$. Hence, we have that

$$\phi(p-1) = \phi(\frac{p-1}{2}) \leq \frac{p-3}{2}.$$

As a result, the probability of a random element $g \in \mathbb{Z}_p$ being a generator of $\mathbb{Z}_p^*$ is given by $d = \frac{\phi(p-1)}{p-1}$, where

$$\frac{1}{6 \ln b} < d \leq \frac{p-3}{2(p-1)}.$$

To deduce the number of trials we would expect to make for our group to find a suitable prime p such that each of our names is a generator of $\mathbb{Z}_p^*$, we first notice that the number of primes smaller than or equal to x is given by $\Pi(x) \approx \frac{x}{\ln x}$ by the Prime Number Theorem, and so the probability that a number p randomly selected from $\{i : 1 \leq i \leq x, i \text{ is odd}\}$ is prime is $\frac{2}{\ln x}$. In particular, we have that

$$\text{Prob[a } b\text{-bit random odd number is prime]} = \frac{2}{\ln 2^b} = \frac{2}{\frac{\lg 2^b}{\lg e}} = \frac{2 \lg e}{b}.$$

Now, given a b-bit prime p, the probability of a random element $g \in \mathbb{Z}_p$ being a generator of $\mathbb{Z}_p^*$ is d. Assuming that our names are random and independent, the probability that each of our names is a generator of $\mathbb{Z}_p^*$ is given by d^s. Hence, the number of trials we would expect to make for our group to find a suitable prime p such that each of our names is a generator of $\mathbb{Z}_p^*$ is

$$\frac{1}{\frac{2 \lg e}{b} \times d^s} = \frac{b}{2d^s \lg e}.$$

(c) Find a prime

Here is the solution from the student group:

Because of the random nature of our program, we ran it several times to get a better estimate of the number of random numbers we would need to try before obtaining a suitable prime.

On average, 28,432 random numbers were tried before we found a suitable prime.

Run 1 names_generate("Soyini Kenny Eduardo Jonathan", 64): Each name in (Soyini Kenny Eduardo Jonathan) is a generator of: 17,561,978,140,628,705,999.

12,628 random numbers were tried before we obtained this result.

Run 2 names_generate("Soyini Kenny Eduardo Jonathan", 64): Each name in (Soyini Kenny Eduardo Jonathan) is a generator of: 16,301,950,415,158,093,319.

27,639 random numbers were tried before we obtained this result.

Run 3 names_generate("Soyini Kenny Eduardo Jonathan", 64): Each name in (Soyini Kenny Eduardo Jonathan) is a generator of: 3,765,661,111,182,547,667.

45,027 random numbers were tried before we obtained this result.

(d) Finding co-Sophie-Germain primes

Here is the solution from the student group:

By the Prime Number Theorem, the probability that a random b-bit number is prime is given by

$$\frac{\Pi(2^b)}{2^b} = \frac{1}{2^b} \cdot \frac{2^b}{\ln 2^b} = \frac{1}{\frac{\lg 2^b}{\lg e}} = \frac{\lg e}{b}.$$

Suppose we randomly choose a 1024-bit number q. The probability that q is a prime is therefore $\frac{\lg e}{1024}$. In order for q to be a Sophie-Germain prime, we also need $2q+1$ to be prime. Note that $2q+1$ would be a 1025-bit number, so the probability that $2q+1$ is prime is given by $\frac{\lg e}{1025}$. Assuming that the likelihood pf q being prime and $2q+1$ being prime is independent, the probability that a random 1024-bit number is a Sophie-Germain Prime is thus $\frac{\lg e}{1024} \times \frac{\lg e}{1025}$. Hence, the number of 1024-bit Sophie-Germain primes we would expect to exist is given by

$$\left(\frac{\lg e}{1024} \times \frac{\lg e}{1025}\right) \times 2^{1024} = \frac{2^{1024} \lg^2 e}{1049600}.$$

Contributors
Solution Design: (entire group effort) - Kenny Fong, Jonathan Hunt, Soyini Liburd, Eduardo McLean.
Solution Implementation: Soyini Liburd.
Write Up: Kenny Fong, Soyini Liburd.
Code Design and Implementation: Soyini Liburd, Eduardo McLean.
Proof Reading: (entire group effort) - Kenny Fong, Jonathan Hunt, Soyini Liburd, Eduardo McLean.
Acknowledgement: the TAs

Problem 3-2. A MAC based on a block cipher

There were many attacks possible on this MAC scheme. Here we present a couple of excellent solutions. The first solution below comes from Stephen Boyer, Xiang Chen, Joseph Corral, and Carolyn Ng.

Using a chosen-plaintext attack on Alice's encryption scheme, we are able to successfully produce new ciphertexts not identical to any seen before that Bob would accept as valid. We achieve this by performing the following steps:

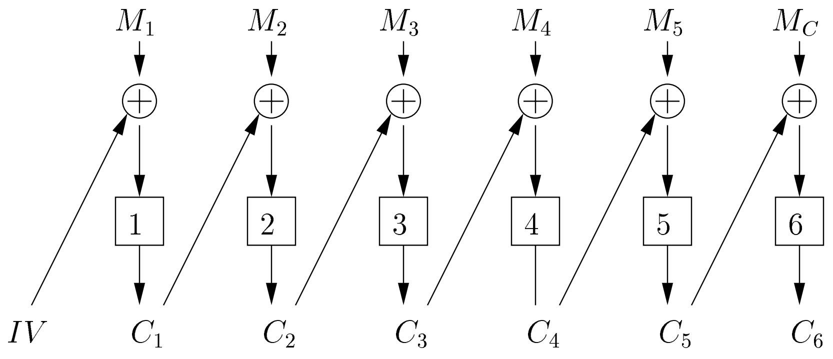
1. We provide Alice with the plaintext to encrypt of the form $M_t = M_1 + M_2 + M_2$ where each M_i is a 128-bit block (we repeat the last block). Here + denotes concatenation.
2. Alice will then produce the ciphertext C_t using AES in the following way: $C_t = IV_1 + C_1 + C_2 + C_3 + C_4$.
3. We then construct a new ciphertext to Bob under the form $C_{\text{forged}} = (IV_1 \oplus M_1) + C_1 + C_2 + C_3$.
4. Bob then decrypts the message which, when verified, is the valid message $0 + M_2$ with IV $IV_1 \oplus M_1$ and checksum M_2.
5. To verify the checksum, Bob computers $0 \oplus M_2$, which is just M_2 so he trusts the forged message.

Solution: Stephen Boyer, Xiang Chen Written by: Stephen Boyer Proofread by: Entire Group

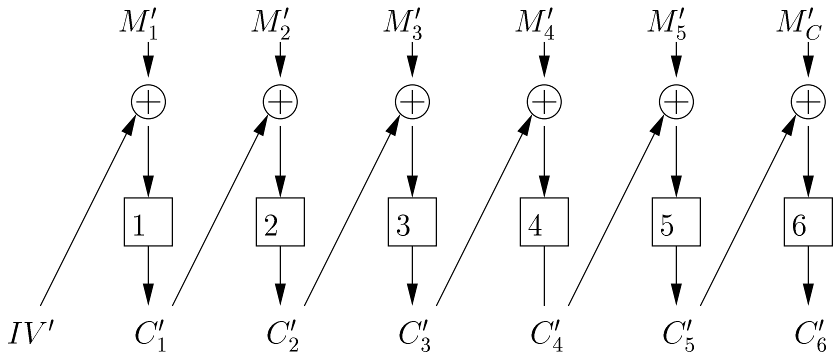
The following solution comes from Jim Paris, David Mellis, Chris Elledge, and Hernan Lombardo.

We can get Alice to sign two messages M and M', and then splice these two messages together to produce a new message M'' which Bob will see as authentic. The splice scrambles one block at the beginning of the message and one block at the splice point, but leaves the data otherwise intact. To do this, consider the following, without loss of generality:

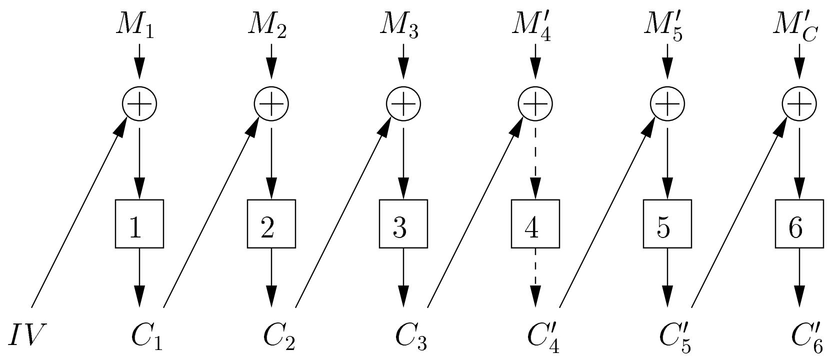
Alice's scheme takes plaintext $\{M_1, M_2, \ldots, M_n\}$ and produces ciphertext $\{C_1, C_2, \ldots, C_{n+1}\}$. The process looks like this, where IV is random and $M_C = M_1 \oplus M_2 \oplus \ldots \oplus M_n$. (The AES steps are numbered for clarity; they are all identical operations utilizing some unknown key K).



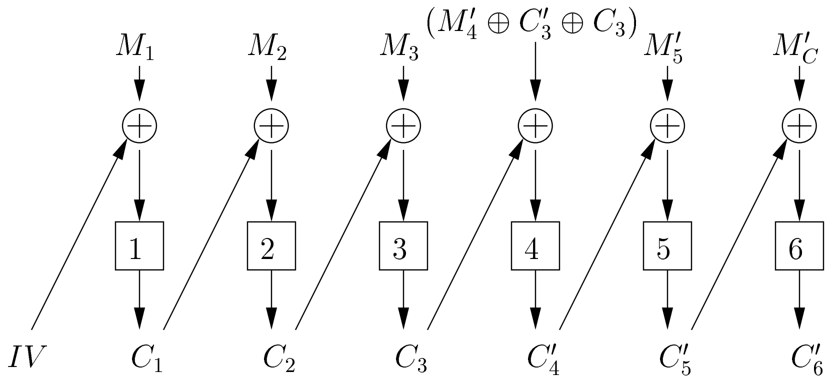
Now assume we just sent her M, and so we now have C as well as IV, and we can compute M_C easily. In other words, we know all of the variables shown in the diagram. Now send a new message M' to get a second set:



Let's now attempt to merge these ciphertexts: let's say that we want to send a message M'' to Bob that consists of $M_1, M_2, M_3, M'_4, M'_5$. Looking at the diagram for this encryption, we have something like this:



This is not fully correct; the dotted line through encryption 4 indicates that the result is *not* C'_4 as desired. However, we can fix this. We know that $E(C'_3 \oplus M'_4) = C'_4$, and so we can replace M'_4 with $(M'_4 \oplus C'_3 \oplus C_3)$ so that encryption 4 gets the same input as before and once again produces C'_4. The encryption now looks like this:



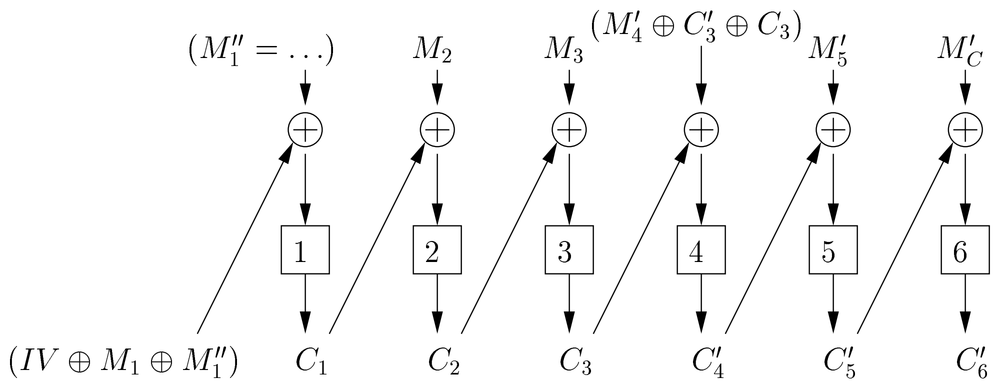
Now the ciphertext properly corresponds to the message blocks listed. However, the checksum M'_C is not correct for this message. Since we can't change the checksum without changing the ciphertext, we instead note that bits in M_1 and IV are interchangeable for the purposes of encryption, and we can therefore change M_1 as necessary to *force* the checksum of the entire message to equal M'_C. That is, we want

$$M'_C = M''_1 \oplus M_2 \oplus M_3 \oplus (M'_4 \oplus C'_3 \oplus C_3) \oplus M'_5$$

which we get by setting

$$M''_1 = M'_C \oplus M_2 \oplus M_3 \oplus (M'_4 \oplus C'_3 \oplus C_3) \oplus M'_5$$

To keep C_1 unchanged, we need $IV'' \oplus M''_1 = IV \oplus M_1$, which we get by setting $IV'' = IV \oplus M_1 \oplus M''_1$. The final diagram looks like this:



To summarize: each encryption operation works on the same input as it did in either the first or second message sent to Alice, and so we know it will generate the ciphertext blocks shown. We send Bob the encrypted message

$$\begin{cases} IV'' = IV \oplus M_1 \oplus M_1'' \\ C'' = \{C_1, C_2, C_3, C_4', C_5', C_6'\} \end{cases}$$

and he will decrypt it into the messages shown in the previous diagram. The checksum is correct, so Bob will assume that the message is valid and was generated by Alice, but it wasn't.

Since none of the operations relied on block counts or the location of the splice, this method can trivially be extended to splice two messages of any length at any point, and can easily be used again to splice more than two messages.

Finally, a much simpler answer to this question: Send Alice $\{M_1, M_2, M_1 \oplus M_2\}$, and she will reply with $\{IV, C_1, C_2, C_3, C_4\}$. Send Bob $\{IV, C_1, C_2, C_3\}$ and he will consider it a valid message even though it's not what Alice returned.

(David Mellis came up with the basic idea, and Jim Paris, Chris Elledge, and Hernan Lombardo helped formalize it into this answer. Jim Paris wrote it up.)

Problem 3-3. Block ciphers need to be non-affine

There were quite a few common errors throughout the problem. The most common mistakes/inexactitues are summarized in *Notes*.

Note: TAES (like *AES*) is a family of functions indexed by a key. In this problem, we study the breaking of *one* (*any* one) function in the family.

I - TAES is an affine transformation

Note: In linear algebra, we always represent vectors as column vectors, so that *all* linear transformations can be represented by matrices (and vice versa). You could also work with the vectors represented by 4 x 4 arrays as in [1], but it would be much harder to capture *all* linear transformations with matrices.

(I-a)

The domains and codomains (a.k.a. ranges) of *ShiftRow*, *MixColumn*, *AddRoundKey*, and $TAES$ are the set of all possible states: $\mathcal{S} = (\mathcal{GF}(2^8))^{16}$. Indeed, these transformations operate on states to give other states. As defined in [1], a state is composed of four rows of four bytes (for the 128-bit version), each byte being an element of the field $\mathcal{GF}(2^8)$. Reordering those elements in a column, we get a column vector of dimension 16.

We can reorder the elements in two orders:
- By row first:
  $(a_{0,0}, a_{0,1}, a_{0,2}, a_{0,3}, a_{1,0}, \ldots, a_{3,3})$
- By column first (which would be the natural order since AES follows this order):
  $(a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, \ldots, a_{3,3})$

In the following, we give the matrices for the two conventions (A for the row-first order, A' for the column-first order).

1.*ShiftRow* is a linear transformation ($b = 0$) where:

$$A = \left(\begin{array}{cccccccccccccccc}
1&0&0&0&&&&&&&&&&&&\\
0&1&0&0&&&&&&&&&&&&\\
0&0&1&0&&&&&&&&&&&&\\
0&0&0&1&&&&&&&&&0&&&\\
&&&&0&1&0&0&&&&&&&&\\
&&&&0&0&1&0&&&&&&&&\\
&&&&0&0&0&1&&&&&&&&\\
&&&&1&0&0&0&&&&&&&&\\
&&&&&&&&0&0&1&0&&&&\\
&&&&&&&&0&0&0&1&&&&\\
&&&&&&&&1&0&0&0&&&&\\
&&&&&&&&0&1&0&0&&&&\\
&&&0&&&&&&&&&0&0&0&1\\
&&&&&&&&&&&&1&0&0&0\\
&&&&&&&&&&&&0&1&0&0\\
&&&&&&&&&&&&0&0&1&0
\end{array}\right), A' = \left(\begin{array}{cccccccccccccccc}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0
\end{array}\right)$$

2.*MixColumn* is a linear transformation ($b = 0$) where:

$$A = \begin{pmatrix} 2*I_4 & 3*I_4 & 1*I_4 & 1*I_4 \\ 1*I_4 & 2*I_4 & 3*I_4 & 1*I_4 \\ 1*I_4 & 1*I_4 & 2*I_4 & 3*I_4 \\ 3*I_4 & 1*I_4 & 1*I_4 & 2*I_4 \end{pmatrix}, A' = \begin{pmatrix} M & 0 & 0 & 0 \\ 0 & M & 0 & 0 \\ 0 & 0 & M & 0 \\ 0 & 0 & 0 & M \end{pmatrix}$$

where I_4 is the identity matrix of dimension 4, and M the *MixColumn* matrix defined in [1]:

$$I_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \text{and } M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

3.*AddRoundKey* is a affine transformation where $A = I_{16}$ is the identity matrix of dimension 16, and b is the round key (as the bitwise XOR is the addition in the field $\mathcal{GF}(2^8)$).

(I-b)

Let $f : \left\{ \begin{array}{lll} E & \to & F \\ x & \mapsto & A_1 * x + b_1 \end{array} \right.$ and $g : \left\{ \begin{array}{lll} F & \to & G \\ y & \mapsto & A_2 * y + b_2 \end{array} \right.$

Then $\forall x \in E, g \circ f(x) = g(f(x)) = A_2 * (A_1 * x + b_1) + b_2 = (A_2 * A_1) * x + (A_2 * b_1 + b_2)$.

Therefore, $g \circ f$ is an affine transformation with $A = A_2 * A_1$ and $b = A_2 * b_1 + b_2$.

By obvious induction, the composition of *any* (finite) number of affine transformations is also affine.

As TAES is the composition of a finite number of *ShiftRow*, *MixColumn*, *AddRoundKey* transformations, TAES is also affine.

Note: The number of such transformations is *not* 3, since we have many rounds.

(I-c)

$TAES$ being affine, $\exists A, b$ such that $\forall x \in \mathcal{S}, TAES(x) = A * x + b$.

We know $TAES$ is invertible (as the general AES is, since we can encrypt/decrypt uniquely).

Note: First, we *need* to prove that the matrix A is invertible. Let us prove it by contradiction.

> If A were not invertible, then A would not be 1-1. Which means that we could find $x_1 \neq x_2$ such that $A * x_1 = A * x_2$. Then, we would have $A * x_1 + b = A * x_2 + b$, i.e. $TAES(x_1) = TAES(x_2)$, and $TAES$ would not be 1-1, thus not invertible.
>
> $\square$

Now, A being invertible, we have:

$$\begin{array}{rrcl} & TAES(x) & = & y \\ \Leftrightarrow & A * x + b & = & y \\ \Leftrightarrow & A * x & = & y - b \\ \Leftrightarrow & x & = & A^{-1}(y - b) \\ \Leftrightarrow & x & = & A^{-1} * y + (-A^{-1} * b) \end{array}$$

Therefore, $TAES^{-1}$ is affine too.

II - An affine transformation is easily breakable

In the following, for convenience, all indices start at 1, not 0.

(II-a)

We just need one pair, if $x = 0$. Indeed, since $g(0) = b$, knowing the pair (0, g(0)) enables us to recover b, and reduces the resolution of $g(x) = A * x + b$ to that of $f(x) = A * x$.

(II-b)

Since A has $n.n = n^2$ unknown coefficients, and each pair $(x, f(x))$ yields n linear equations, we need at least n pairs.

Let us have a closer look at the n linear equations. Knowing the pair $(x^i, y^i = f(x^i))$ yields the following linear equations (the superscript here is used as an index, and *not* with the standard power meaning):

$$(R_i)\left\{\begin{array}{ccccccc} A_{1,1}*x_1^i & + & A_{1,2}*x_2^i & +\cdots+ & A_{1,n}*x_n^i & = & y_1^i \\ A_{2,1}*x_1^i & + & A_{2,2}*x_2^i & +\cdots+ & A_{2,n}*x_n^i & = & y_2^i \\ \vdots & + & \vdots & +\ddots+ & \vdots & = & \vdots \\ A_{n,1}*x_1^i & + & A_{n,2}*x_2^i & +\cdots+ & A_{n,n}*x_n^i & = & y_n^i \end{array}\right.$$

Thus, knowing n pairs $(x^1, y^1 = f(x^1)), \ldots, (x^n, y^n = f(x^n))$ yields n systems of n linear equations (we take the j^{th} equation given by each pair to form (S_j)):

$$\forall j \in \{1,2,\ldots,n\}, (S_j)\left\{\begin{array}{ccccccc} A_{j,1}*x_1^1 & + & A_{j,2}*x_2^1 & +\cdots+ & A_{j,n}*x_n^1 & = & y_j^1 \\ A_{j,1}*x_1^2 & + & A_{j,2}*x_2^2 & +\cdots+ & A_{j,n}*x_n^2 & = & y_j^2 \\ \vdots & + & \vdots & +\ddots+ & \vdots & = & \vdots \\ A_{j,1}*x_1^n & + & A_{j,2}*x_2^n & +\cdots+ & A_{j,n}*x_n^n & = & y_j^n \end{array}\right.$$

Using matrix form:

$$\forall j \in \{1,2,\ldots,n\}, (S_j)\begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_n^1 \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^n & x_2^n & \cdots & x_n^n \end{pmatrix} * \begin{pmatrix} A_{j,1} \\ A_{j,2} \\ \vdots \\ A_{j,n} \end{pmatrix} = \begin{pmatrix} y_j^1 \\ y_j^2 \\ \vdots \\ y_j^n \end{pmatrix}$$

Each system (S_j) is a system of n linear equations with n unknowns $A_{j,1}, A_{j,2}, \cdots, A_{j,n}$. A necessary and sufficient condition for the system (S_j) to have a uniquely determined solution is to have a non-null determinant for the matrix $(x_k^i)_{i,k}$, i.e. that the $(x^1, x^2, \ldots, x^n)$ be linearly independent. Note that all systems (S_j) have the same $(x_k^i)_{i,k}$ matrix, but with different $(y_j^k)_k$ constants.

In conclusion, we just need n pairs $(x^i, y^i = f(x^i))$ if the $(x^1, x^2, \ldots, x^n)$ are linearly independent.

Note: Considering all the equations as a system of n^2 linear equations with n^2 unknowns would have forced us to relate the non-nullity of the n^2 x n^2 determinant to the linear independence of the $(x^1, x^2, \ldots, x^n)$, which is a bit less straightforward.

III - How to break TAES

By part I, we know that $TAES$ is affine. Therefore, $\exists A, b$ such that $\forall x \in \mathcal{S}, TAES(x) = A * x + b$.

We first query the null message $m_0 = 0$, so that we get $b = TAES(0)$.

Then, for $i \in \{1, 2, \ldots, 16\}$, we query the message $m_i = 0^{2*(i-1)}||01||0^{2*(16-i)}$ in hexadecimal: $(m_1, m_2, \cdots, m_{16})$ is the unit basis of the vector space $\mathcal{S}$.

Now we can compute the columns $(A_1, A_2, \ldots, A_{16})$ of A as follows: $A_i = TAES(m_i) - TAES(m_0)$.

We only needed to compute 16 matrix "additions"!!!

In conclusion, we only need 17 plaintext queries to entirely break TAES, i.e. encrypt any message we want afterwards.

Of course, then, by inverting the matrix A, we can also decrypt any message we want.

Remark: A similar ciphertext attach would have given us $TAES^{-1}$ without much computation, and then one matrix inversion would have given us $TAES$.

Last but not least: Two groups (Jim Paris, David Mellis, Chris Elledge, Hernan Lombardo & Levente Jakab, Josh Marron, Paul Youn, Enrique Zolezzi) actually found better: you actually just need one *one* plaintext (or ciphertext) query. Bravo!!!

A careful study of the details of the algorithm show that the matrix A is the result of the multiplication of the matrices of *all* the individual components (over *all* the rounds), and thus does *not* depend on the key. In other words, A is constant (and publicly known) for the whole $TAES$ family (since the algorithm is public). The only unknown is thus the vector b, the recovering of which needs only *one* plaintext (or ciphertext) query.

References

[1] Joan Daemen and Vincent Rijmen. Rijndael: the Advanced Encryption Standard. *Dr. Dobb's Journal*, pages 137–139, March 2001.