

Suitable function

Daniel Nager
daniel.nager@gmail.com

February 10, 2021

This file and referenced files are on the address <https://github.com/danielnager/xifrat/>

We will work with 4x4 anti-circulant matrices of the form:

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_1 & a_2 & a_3 & a_0 \\ a_2 & a_3 & a_0 & a_1 \\ a_3 & a_0 & a_1 & a_2 \end{pmatrix}$$

With $a_i \in \mathbb{Z}_p$ with p a large enough prime, say 32-bit. Let's call the invertible subset of such matrices S .

Since the product of two anti-circulant matrices is a circulant matrix, we define a matrix, that turns a circulant matrix into its anti-circulant equivalent.

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Next we define a function f , that's the product of two matrices turning the result in its anti-circulant equivalent:

$$f(A, B) = ABM$$

As an anticirculant matrix is defined by its first column, we can alternatively write f just as a function of vectors:

$$f\left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}, \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}\right) = \begin{pmatrix} a_0b_0 + a_1b_1 + a_2b_2 + a_3b_3 \\ a_0b_1 + a_1b_2 + a_2b_3 + a_3b_0 \\ a_0b_2 + a_1b_3 + a_2b_0 + a_3b_1 \\ a_0b_3 + a_1b_0 + a_2b_1 + a_3b_2 \end{pmatrix}$$

The following two properties hold:

$f(A, B) \neq f(B, A)$ – non-commutativity in general

$f(f(A, B), C) \neq f(A, f(B, C))$ – non-associativity in general

$f(f(A, B), f(C, D)) = f(f(A, C), f(B, D))$ – restricted commutativity

Next we define a list of N elements of such matrices to meet the size required. For 256 bits we can use $p \approx 2^{32}$, and $N = 2$.

Next we define a mixing procedure of elements of this kind, t and k , N -element lists of elements in S .

The mixing procedure is:

```
function m(t,k) returns t

    for M number of rounds -- 64 for example
        //one-to-one mixing of k and t
        for i in 0..N-1
            t[i]<-f(t[i],k[i])
        end for
        // accumulative mixing of t with itself, t[-1]=t[N-1]
        for i in 0..N-1
            t[i]<-f(t[i],t[i-1])
        end for
    end for

    return t
```

The function m is neither associative nor commutative, and meets the restricted commutativity property:

$$m(m(a, b), m(c, d)) = m(m(a, c), m(b, d))$$

With this a Secret agreement and a Digital signature can be done as explained in the document:

<https://github.com/danielnager/xifrat/blob/raw/cryptosystem.pdf>

The computationally hard problem proposed is:

in $c = m(t, k)$, knowing c and t , find k .

Now let's define the secret agreement and the digital signature using the mixing function m . To put it more clear we will use the following notation:

$m(a, b)$ is written as (ab)

$m(m(a, b), m(c, d))$ is written as $(ab)(cd)$

$m(m(a, b), c \dots)$ is written as $(abc \dots)$

For the secret agreement the procedure is the following:

Both Alice and Bob agree on some constant C . Alice chooses a random key K , and Bob does the same choosing a random key Q . Alice sends to Bob (CK) , Bob sends to Alice (CQ) . Alice computes using Bob's sent value $(CQ)(KC)$, and Bob does the same and computes $(CK)(QC)$.

By the property of restricted commutativity $(CQ)(KC) = (CK)(QC)$

For the signature the procedure is the following:

Alice, the signer, chooses a public value C and two random keys K, Q . Its credentials are C , (CK) and (QK) . To sign a value, H , Alice computes $S = (HQ)$.

Bob needs to verify if Alice has signed H . Computes $(HQ)(CK)$ and $(HC)(QK)$. Both values must be equal due to restricted commutativity if (HQ) is a valid signature from Alice.

In order to do smaller signatures, of 128 bits in this case, there's an approach that must be carefully tested.

We apply the following equality:

$$(QCK)(KH_1H_2Q) = (QK)(CH_1)(CH_2)(KQ)$$

In this formula, C is 128 bit public constant provided by Alice, the signer, K and Q are two 128 bit keys known only by the signer, and H_1 and H_2 is a 256 bits value to be signed split in two halves.

The credentials of Alice are (QCK) , (QK) and (KQ) .

In order to sign a value represented by H_1 and H_2 , the Alice computes $S = (KH_1H_2Q)$.

To verify the signature Bob computes (CH_1) and (CH_2) , and checks for the initial equality to hold, as Bob has all the elements needed. If the equality holds then is a valid signature from Alice.

Daniel Nager - daniel.nager@gmail.com