Secret agreement and Digital signature

Daniel Nager
daniel.nager@gmail.com

March 8, 2021

Summary

In this document a cryptographic theoretical primitive is presented, among with two public key protocols, one of secret agreement and another of digital signature. The primitive does not depend on the problem of discrete logarithm or the factorization. Therefore, the usual approaches related to the solution of the discrete logarithm cannot be applied.

Key and constant sizes are reduced. The signature, for example, requires 256 bits to achieve 128 bits of security. Another sizes are possible.

The document includes three parts. First, a description of the primitive and protocols proposed, second, a brief security analysis, and finally a section of proofs.

Cryptographic structure description

Basic function

We need to define a function in order to use it as a basis for the round construction. The function takes two parameters, $f(B, A)$, A and B belong to a given set with some properties. Let's call this set S. Given that the A parameter has a different function than the B one, f function can be written as $f_A(B) = f(B, A)$, depending on the context.

The function f must be non-associative, so $f(f(A, B), C) \neq f(A, f(B, C))$ in general terms. The function f can be commutative or not. The important property needed is, we can call, restricted commutativity, i.e. $f(f(A, B), f(C, D)) = f(f(A, C), f(B, D))$. In the left and right parts of the equality B and C can be swapped given raise to the same result.

The f function with the second parameter fixed, $f_A(B)$, must be a permutation, so $f_A(B) = f_A(C) \Leftrightarrow B = C$.

Mixing procedure and cryptographic elements

Independently of the particular definition of f, we can define a mixing procedure m of the elements that form the keys and constants or values to sing that works only if the f function meets the requisites of the previous section. Here we prove that if $f_A(B)$ is a cryptographic permutation that meets the restricted commutativity, the same is extended to the mixing result of m.

We will work with lists of N elements of S depending of the size in bits of the elements of S and the desired security level. So, if S contains n-bit elements we need $N = 256/n$, N elements in the lists produces a 256 bit output.

Cryptographic elements

Once fixed N, the size of the list of elements of S, which serves as parameters and result of the m function, we define L as the set of lists of length N of elements in S.

In the final result we will end in a mixing function $R = m(T, K)$, where R, T and K will be lists of N elements of S, so $R \in L$, $T \in L$, $K \in L$ are lists of N elements of the S set. If E is such a list E_i will be an element of this list, in the range $[0, N-1]$. We define also $E_{-1} = E_{N-1}$ for the sake of simplicity.

Mix function

We define a mix function $R = m(T, K)$, which must be non-associative and not commutative meet the restricted commutativity, while not being necessarily commutative, so $m(m(A, B), m(C, D)) = m((A, C), m(B, D))$, besides $m_K(T)$ is a cryptographic permutation.

The mix function is defined with the following procedure:

function $r(T, K)$
  for i **from** 0 **to** $N - 1$ **do**
    /// initial mixing of K and T
    $R_i \leftarrow f(T_i, K_i)$
  end for

  $e \leftarrow first\_seq$
  for i **from** 0 **to** $M - 1$ **do**
    /// mixing of R with itself in random sequence
    $d \leftarrow next\_seq$
    $R_i \leftarrow f(R_d, R_e)$
    $e \leftarrow d$
  end for

  for i **from** 0 **to** $N - 1$ **do**
    /// final mixing of K and R
    $R_i \leftarrow f(R_i, K_i)$
  end for
  return R
end function

This function is non-associative, non-commutative and meets the restricted commutativity (*proof 1*). If we fix one of the two parameters of r, T or K we get a cryptographic permutation of the other parameter, since in one round you can recover K knowing T, or T knowing K, if f is a permutation as well.

The values returned by *next_seq* are positions from 0 to $N - 1$, and are always returned in the same sequence, *first_seq* returns the first element of the sequence.

We need to define how many rounds are needed, which depends on the f function and its complexity. In any case they aren't a lot. $M = 64 \cdot N$ is a reasonable value, being possible to justify less or more rounds, depending on the actual function f.

This function m, is neither commutative nor associative, besides meeting the mentioned property of restricted commutativity, so, and this is important in order to define the protocols of secret agreement and digital signature:

$m(m(A,B),m(C,D)) = m(m(A,C),m(B,D))$, where A, B, C and D are lists of N elements in the set S.

Fixing T or K in m we get, as in the round function, a cryptographic permutation of K or T accordingly. As this is shown for r, m is a concatenation of r, so a permutation of a permutation for V rounds, that is a permutation. This result means there aren't related keys. Every key produces a different permutation on a fixed plaintext.

Computationaly intractable problem

Once defined the m function, the computationally intractable problem is, in the equality $R = m(T,K)$, knowing R and T, find the corresponding K. This will be developed in more detail at the initial security analysis.

Secret agreement and digital signature protocols

Once defined the cryptographic primitive m, operating with two elements of L, lists of N elements in the set S, in such a way that $m_A(B)$ is a cryptographic permutation, m is not commutative and not associative but meets the property $m(m(A,B),m(C,D)) = m(m(A,C),m(B,D))$, we can define both procedures, one to share a secret and another to credit a digital signature.

Let's call further on the sides of communication over an unsafe channel as A and B, not to be confused with the previous use of A and B in the properties definition.

Secret agreement

The procedure to end with a shared secret is the following:

A and B share a public constant value C, $C \in L$.

A chooses a secret key K, $K \in L$ and sends to B: $A_I = m(C, K)$

B chooses a secret key Q, $Q \in L$, and sends to A: $B_I = m(C, Q)$

A computes $A_J = m(K, C)$ and operates it with B_I, $S_A = m(B_I, A_J)$

B computes $B_J = m(Q, C)$ and operates it with A_I, $S_B = m(A_I, B_J)$

By the property of restricted commutativity we have that $S_A = S_B$, since expanding the values of S_A and S_B we have $m(m(C, K), m(Q, C)) = m(m(C, Q), m(K, C))$. Let's note that m is not commutative, and so $m(K, C) \neq m(C, K)$ and $m(Q, C) \neq m(C, Q)$. This way, $S = S_A = S_B$, cannot be inferred from K, Q, A_I or B_I.

Digital signature

The procedure to do a digital signature, including the credential definition and the signature itself, is the following:

The signer first chooses two keys $K, Q \in L$, which must be kept secret. Q will be used for signing the required values. Also chooses a public constant C, $C \in L$. Then computes two public values $P = m(C, K)$ and $R = m(Q, K)$.

Now, the only signer secret that must be keept is Q, and its public credentials are C, P and R.

To sing a value H, $H \in L$, the signature S, is $S = m(H, Q)$.

The verifier must compute $V = m(H, C)$, and check the following equality holds:

$$m(S, P) = m(V, R)$$

which is the same as verifying, expanding the formula, that:

$$m(m(H, Q), m(C, K)) = m(m(H, C), m(Q, K))$$

which is true due to the restricted commutativity property if the signature is valid.

Initial security analysis

1 f function is not associative. This means that while rounds of mixing are done, the complexity increases, without possibility of reduction as brackets cannot be removed. Otherwise, the result of all the mixing process will be a simple formula with powers of the initial elements, and so vulnerable to attacks. Non-associativity prevents this.

2 Algorithms related to solutions or methods to improve the calculation of discrete logarithm are not appliable here. The mixing process prevents this if we're not working with a perfectly ordered cyclic group with an exponentiation of a generator.

3 Security relies also on the robustness of the basic f function, there are functions meeting all the requirements presented here that led to absolutely unsafe cryptosystems.

Proofs

Proof 1

To do this proof we need to use the result of *proof 3*.

We need to prove that $R = m(m(A,B), m(C,D)) = m(m(A,C), m(B,D))$.

Let's define $a = [a_i]$, $b[b_i]$, $c = [c_i]$, $d = [d_i]$, $R = [r_i]$ lists of elements of S.

First of all, let's simplify the notation.

$f(a,b)$ is written just as ab, we will use ordinary product as f.

$(f(f \ldots (a,b), c \ldots$ is written as $abc \cdots$. As f is not associative f will be applied from left to right in series of products.

Finally we state that given to lists a and b $[a_i][b_i] = [a_i b_i]$. We operate two elements at the same position to give a result in the same position.

Next, given a list $[a_i]$, we define a function e_k that represents a given structure of nested combinations of arbitrary a_i applying f.

For example $e_k([a_0, a_1, a_2]) = (a_1(a_2 a_1)a_3 a_0)$

Next, we define a general e function that operates on a list of lists, so $e(a) = [e_i(a)]$. This e characterizes the round part of the mixing function.

To further prove, let's state the following property:

$e(ab) = e(a)e(b)$. a and b are lists and e a particular nested structure.

With all those properties we can prove the initial assertion

From m, the mixing function, we assume a fixed nested structure of $a_i b_i$, ignoring the last key mixing. This structure is e.

So $m(a,b)$ results in $e(ab)b$, and $m(c,d)$ in $e(cd)d$, adding the final mixing of the initial second parameter. Let's recall that a, b, c, d are lists of elmenents in s.

Lets proceed to mix $m(a,b)$ with $m(c,d)$, and $m(a,c)$ with $m(b,d)$:

$$m(m(a,b), m(c,d)) = (e(e(ab)b))(e(e(cd)d))(e(cd)d)$$
$$m(m(a,c), m(b,d)) = (e(e(ac)c))(e(e(bd)d))(e(bd)d)$$

Let's apply the property $e(ab) = e(a)e(b)$ to both right sides of those equations. We have:

$$\begin{gathered}(e(e(a))e(e(b))e(b))\ \ (e(e(c))e(e(d))e(d))\ \ (e(c)e(d)d) \\ = \\ (e(e(a))e(e(c))e(c))\ \ (e(e(b))e(e(d))e(d))\ \ (e(b)e(d)d)\end{gathered}$$

From *proof 3* $(e_0e_1e_2)(e_3e_4e_5)(e_6e_7e_8) = (e_0e_3e_6)(e_1e_4e_7)(e_2e_5e_8)$ and thus we can interchange lists c and b in the previous formula, so the initial proposition is correct.

Restricted commutativity extension - proof 3

Basic operator definition

We define an operator $\star$, based on some underlying set and satisfying the following property, named restricted commutativity:

$$(a \star b) \star (c \star d) = (a \star c) \star (b \star d)$$

From left to right sides of the equality b and c are interchanged.
An example of such an operator can be:

$$a \star b = ab^2$$

Then:

$$(a \star b) \star (c \star d) = ab^2(cd^2)^2 = ab^2c^2d^4 = ac^2b^2d^4 = (a \star c) \star (b \star d)$$

Nomenclature

In order to simplify, for now on the operator $\star$ will be represented as an space, and the order of application will be from left to right in absence of explicit brackets. Then:

$$a \star b \text{ is written as } ab$$

$$(abcd\dots) \text{ means } ((((ab)c)d)\dots)$$

Property proof

We need to prove that inside a set of equal sized tuples, we can rearrange elements in a given way and get the same final result of operation the tuples either way, provided the operator used meets the restricted commutativity property.

An example of this tuple equality is:

$$(abc)(def)(ghi)(jkl) = (adgj)(behk)(cfil) = r$$

$2 \times n$ series of tuples

Starting with:

$$(a_1a_2 \dots a_{n-1}a_n)(b_1b_2 \dots b_{n-1}b_n)$$

we can add brackets:

$$((a_1a_2 \dots a_{n-1})a_n)((b_1b_2 \dots b_{n-1})b_n)$$

and by definition of the restricted commutativity:

$$((a_1a_2 \dots a_{n-1})(b_1b_2 \dots b_{n-1}))(a_nb_n)$$

this reduces the problem one step. By induction we get:

$$(a_1a_2 \dots a_n)(b_1b_2 \dots b_n) = (a_1b_1)(a_2b_2) \dots (a_nb_n)$$

In general, $2 \times n$ and $n \times 2$ series of tuples can interchange its values in a given form. The proven formula is:

$$(a_1a_2 \dots a_{n-1}a_n)(b_1b_2 \dots b_{n-1}b_n) = (a_1b_1)(a_2b_2) \dots (a_{n-1}b_{n-1})(a_nb_n)$$

$n \times m$ series of tuples

We need to prove that:

$$(a_{1,1}a_{1,2}\dots a_{1,n})(a_{2,1}a_{2,2}\dots a_{2,n})\dots(a_{m,1}a_{m,2}\dots a_{m,n})$$
$$=$$
$$(a_{1,1}a_{2,1}\dots a_{m,1})(a_{1,2}a_{2,2}\dots a_{m,2})\dots(a_{1,n}a_{2,n}\dots a_{m,n})$$
$$= r$$

Adding brackets to the first term of the equality we get:

$$((a_{1,1}a_{1,2}\dots a_{1,n-1})a_{1,n})((a_{2,1}a_{2,2}\dots a_{2,n-1})a_{2,n})\dots((a_{m,1}a_{m,2}\dots a_{m,n-1})a_{m,n})$$

And using the result for $2 \times n$ tuples we reduce to:

$$((a_{1,1}a_{1,2}\dots a_{1,n-1})(a_{2,1}a_{2,2}\dots a_{2,n-1})\dots(a_{m,1}a_{m,2}\dots a_{m,n-1}))(a_{1,n}a_{2,n}\dots a_{m,n})$$

simplifying the problem one step. By induction we get in the end:

$$(a_{1,1}a_{2,1}\dots a_{m,1})(a_{1,2}a_{2,2}\dots a_{m,2})\dots(a_{1,n}a_{2,n}\dots a_{m,n})$$

Which is the second term of the above equation. This result can be extended to subtuples, inside a more complex formula, as in the following example:

$$(a(bc))(d(ef)) = (ad)((bc)(ef))$$

This result is useful in order to do other proofs on formulas based on an operator that meets the property of restricted commutativity.