

Resurrecting Xifrat - Compact Cryptosystems 2nd Attempt

Xifrat was a group-theoretic public-key cryptosystem based on a quasigroup with the special property of "restricted-commutativity". It was broken within half a month of its publication, due to a mistake made in the "mixing" function. In this paper, we revisit the design decisions made, proposing new constructions, and attempt (again) to build secure digital signature schemes and key encapsulation mechanisms. If the schemes can be proven secure, then this will be the most compact and the most efficient post-quantum cryptosystem ever proposed to date.

Authors: Jianfang "Danny" Niu (dannyniu {at} hotmail {dot} com)

!NOTE! The HTML rendering of this document is not authoritative, and readers seeking a stable reference should look for the PDF version published at official sources.

Table of Contents

1. Background	3
2. Quasigroup and the Blk block function	3
2.1. The restricted commutative quasigroup	4
2.2. The generalized restricted commutativity	4
2.3. The Blk block function	5
3. The Enc and Mlt functions and Xifrat1-Sign	6
3.1. Restricted Commutativity of Mlt over Enc (and vice versa)	6
3.2. The Xifrat1-Sign DSS	7
4. The Vec and Dup functions and Xifrat1-Kex.I	9
4.1. Restricted Commutativity of Vec and Dup over themselves	9
4.2. The Xifrat1-Kex.I KEM	10
Annex A. References	11

Figures

Figure 2.1. The algorithm for the Blk function	6
Figure 3.1. The algorithm for the Enc function	7
Figure 3.2. The algorithm for the Mlt function	7
Figure 3.3. Xifrat1-Sign Key Generation	8
Figure 3.4. Xifrat1-Sign Signature Generation	8
Figure 3.5. Xifrat1-Sign Signature Verification	8
Figure 4.1. The algorithm for the Vec function	9
Figure 4.2. The algorithm for the Dup function	10
Figure 4.3. Xifrat1-Kex.I Key Generation	11
Figure 4.4. Xifrat1-Kex.I Encapsulation	11
Figure 4.5. Xifrat1-Kex.I Decapsulation	11

1. Background

In this paper, we propose 2 group-theoretic compact public-key cryptosystems - a key encapsulation mechanism (KEM) and a digital signature scheme (DSS). These cryptosystems are aimed at achieving at least 192-bit post-quantum security.

Previously, most practical compact cryptosystems are based on elliptic curves, whether it's the pre-quantum ones based on EC discrete logarithm [\[SEC#1\]](#), or elliptic curve isogenies. The former are vulnerable to Shor's algorithm on quantum computers; while the latter are suffering from some performance problems. For example, SIKE [\[SIKE\]](#) is a NIST 3rd-round KEM/PKE alternate candidate that executes in time that's an order of magnitude longer than lattice-based ones such as Kyber [\[Kyber\]](#) and Saber [\[Saber\]](#), although, such difference is not too noticeable; likewise, a later design that didn't manage to get into the NIST PQC project - SQISign [\[SQISign\]](#) from Oct 2020 has signing time that's over 2 seconds long on modern workstation PCs.

Group-theoretic cryptography are, in the opinion of the author, still in its infancy - with closures of various theoretical structures and properties being proposed and analyzed without anything remarkable turning up. Algebraic Eraser [\[AE\]](#) being a prominent example based on braid group for key agreement that had failed to be standardized in an ISO/IEC standard; WalnutDSA [\[WalnutDSA\]](#) being another prominent example that didn't pass the 1st round in the NIST PQC project. Both due to security issues.

Xifrat aims to provide PQC schemes that're compact through use of a class of groupoid with the property of *restricted-commutativity*. Such groupoid was previously proposed in [\[NN21\]](#), however, a critical error was made in designing the "mixing" function, which resulted in a total break, just half a month after its publication. We retroactively name the scheme in that paper Xifrat0 (and Xifrat0-Kex and Xifrat0-Sign). In this paper, we revisit the design decisions, and devise new constructions that can be used securely (or more accurately: *may* be used securely **if** the underlying primitive can be proven secure).

2. Quasigroup and the Blk block function

In this section, we present the quasigroup table, discuss the property of restricted commutativity and its generalization (which we will be using in the PKC schemes), and present a construction that enlarges the quasigroup.

2.1. The restricted commutative quasigroup

The quasigroup we're considering has the following properties:

- Non-Associative *In General*: that is, for most cases, $(ab)c \neq a(bc)$
- Non-Commutative *In General*: that is, for most cases, $ab \neq ba$
- Restricted-Commutativity: that is, for all cases, $(ab)(cd) = (ac)(bd)$

Additionally, some properties are needed for basic security:

- The quasigroup table should overall be not symmetric;
- The quasigroup table should not have any fixed points;

We observed that, in Xifrat0, as well as the [StackExchange post](#) that sparked all these discussion, the quasigroup tables had a regularity that, for each diagonal pair of equal table cells, the opposite diagonal is also equal. This appears to be a necessary but not sufficient condition for a power-of-2 table to be restricted-commutative; as for non-power-of-2 tables, experiment had shown this property does not apply to them.

We used diagonal property for optimization and created a new program that searched for a random quasigroup table with the seed "xifrat - public-key cryptosystem" which is the same one that's used in Xifrat0. Although we had hoped for that our optimization can make it possible to find a 16-by-16 quasigroup, the poly-exponential-time complexity ultimately convinced us to give up.

The source code for the new program can be found at our online git repository: <https://github.com/dannyniu/xifrat> The new table is as follow:

```
// Quasigroup generated using the new program //
 2  0  4  3  5  7  1  6
 1  5  3  4  0  6  2  7
 7  4  0  5  3  2  6  1
 0  2  7  6  1  4  5  3
 3  6  1  2  7  5  4  0
 6  3  5  0  4  1  7  2
 4  7  2  1  6  0  3  5
 5  1  6  7  2  3  0  4
```

The operation ab evaluates to the table cell at a'th row and b'th column, in 0-based index.

We propose the **1st open problem** of this paper: Can we find a verifiably random 16-by-16 quasigroup table? Can we find one efficiently?

2.2. The generalized restricted commutativity

Now we introduce an important property, that is both useful, and comes naturally from restricted-commutativity: the generalized restricted commutativity.

Theorem 1. *Left-associativity of distributiveness*

That is:

$$(a_1 b_1)(a_2 b_2) \dots (a_n b_n) = (a_1 a_2 \dots a_n)(b_1 b_2 \dots b_n)$$

Proof:

Observe a case of 3 pairs: $(ab)(cd)(ef)$.

due to restricted commutativity: $(ac)(bd)(ef)$,

next, substitute $g=(ac)$, $h=(bd)$, we have:

$$(gh)(ef)$$
 ,

again, due to restricted commutativity, we have: $(ge)(hf)$,

substitute back, we have $(ace)(bdf)$,

generalizing recursively, we have **Theorem 1.**

Property 1. *Generalized Restricted-Commutativity*

That is:

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

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

Proof: From **Theorem 1.**, we have

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

$$((x_{1,1} x_{2,1})(x_{1,2} x_{2,2}) \dots (x_{1,n} x_{2,n})) (x_{3,1} x_{3,2} \dots x_{3,n}) \dots (x_{m,1} x_{m,2} \dots x_{m,n}) =$$

$$(((x_{1,1} x_{2,1}) x_{3,1}) ((x_{1,2} x_{2,2}) x_{3,2}) \dots (x_{1,n} x_{2,n} \dots x_{m,n})) =$$

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

2.3. The Blk block function

The Blk block function is defined to enlarge the quasigroup - it operates on vector of 21 trit bitstrings. This is 63-bit in total, which we fit in least-significant- bit&byte -first order.

Figure 2.1. The algorithm for the Blk function

- Input: $A=(a_0 a_1 \dots a_{20})$, $B=(b_0 b_1 \dots b_{20})$
- Output: $C=(c_0 c_1 \dots c_{20})$

Steps:

- $c_0 = (a_0 a_1 \dots a_{20}) (b_0 b_1 \dots b_{20}) (a_0 a_1 \dots a_{20}) (b_0 b_1 \dots b_{20})$
- $c_1 = (a_1 a_2 \dots a_0) (b_1 b_2 \dots b_0) (a_1 a_2 \dots a_0) (b_1 b_2 \dots b_0)$
- $c_2 = (a_2 a_3 \dots a_1) (b_2 b_3 \dots b_1) (a_2 a_3 \dots a_1) (b_2 b_3 \dots b_1)$
- ...
- $c_{20} = (a_{20} a_0 \dots a_{19}) (b_{20} b_0 \dots b_{19}) (a_{20} a_0 \dots a_{19}) (b_{20} b_0 \dots b_{19})$

Programmatically, A , B , and C are represented as the `uint64_t` data type, with the top bit clear.

There is one problem with the Blk function, that is, when it's given 2 vectors of repeated tritets, it produces a vector of repeated tritet. This is due to such input would produce same array of operations at every lane of output tritet. At the moment, we do not know if this is exploitable in the actual KEM and DSS scheme.

We propose the **2nd open problem** of this paper: Is there other more efficient construction which is similar to both Blk and Enc, in that such construction provides key-hiding and is restricted-commutative over itself?

3. The Enc and Mlt functions and Xifrat1-Sign

In this section, we present the Enc and Mlt functions, argue the security property for Enc empirically, and present construction for a digital signature scheme

3.1. Restricted Commutativity of Mlt over Enc (and vice versa)

The Enc function is designed to hide its right-hand operand, that is given P and $C = \text{Enc}(P,K)$, it should be cryptographically impossible to discover K (assuming K is randomly generated). The construction of Enc is inspired by the "sandwich" structure of some of the tweakable blockcipher modes such as XEX (xor-encrypt-xor) and EME (encrypt-mask-encrypt). The way we design the Enc function make it impossible to make it restricted-commutative with itself, so we introduced a second Mlt function, which together with Enc have the following property:

$$M(E(a,b), E(c,d)) = E(M(a,c), M(b,d))$$

As a conservative design decision, we choose to have a 7-slice Feistel-network-like structure for cryptograms for Enc and Mlt, totalling 448 bits (441 effective), aiming at offering more than 192-bit security.

Figure 3.1. The algorithm for the Enc function

- Input: $A=(A_0 A_1 \dots A_6)$, $B=(B_0 B_1 \dots B_6)$
- Output: $C=(C_0 C_1 \dots C_6)$

Steps:

- $C_0 = (\dots((B_0 A_0 B_0) B_1 A_1 B_1) \dots) B_6 A_6 B_6$
 - $C_1 = (\dots((B_1 A_1 B_1) B_2 A_2 B_2) \dots) B_0 A_0 B_0$
 - $C_2 = (\dots((B_2 A_2 B_2) B_3 A_3 B_3) \dots) B_1 A_1 B_1$
 - ...
 - $C_6 = (\dots((B_6 A_6 B_6) B_0 A_0 B_0) \dots) B_5 A_5 B_5$
-

We put parentheses in, but it's just equivalent to the actual operation where all operands are just evaluated in order.

Figure 3.2. The algorithm for the Mlt function

- Input: $A=(A_0 A_1 \dots A_6)$, $B=(B_0 B_1 \dots B_6)$
- Output: $C=(C_0 C_1 \dots C_6)$

Steps:

- $C_i = A_i B_i$, $i \in \{ 0, 1, 2, 3, 4, 5, 6 \}$
-

3.2. The Xifrat1-Sign DSS

Now we present the Xifrat1-Sign digital signature scheme. The general structure is similar to Xifrat0-Sign, but use Enc and Mlt functions to actually achieve unforgeability.

As with Xifrat0-Sign, we use a hash function, which is instantiated with the XOF SHAKE-256. We take its initial 448-bit output, interpret it as 7 64-bit unsigned integers in little-endian, and clear each of their top bits. We denote this hash function as $Hx_{448-7}(m)$.

Figure 3.3. Xifrat1-Sign Key Generation

1. Uniformly randomly generate 3 cryptograms: C , K , and Q ,
2. Compute $P_1 = E(C,K)$, $P_2 = M(K,Q)$,
3. Return public-key $pk = (C, P_1, P_2)$ and private-key $sk = (C, K, Q)$.

Figure 3.4. Xifrat1-Sign Signature Generation

1. **Input:** m - the message
2. Compute $H = \text{Hx}_{448-7}(m)$,
3. Compute $S = E(H,Q)$,
4. Return S ,

Figure 3.5. Xifrat1-Sign Signature Verification

1. **Input:** m - the message, S - the signature
2. Compute $H = \text{Hx}_{448-7}(m)$,
3. Compute $T_1 = M(P_1, S)$,
4. Compute $T_2 = E(M(C,H), P_2)$,
5. If $T_1 = T_2$ return [VALID]; otherwise return [INVALID].

The proof of correctness of the scheme is as follow:

$$T_1 = M(P_1, S) = M(E(C,K), E(H,Q))$$

$$T_2 = E(M(C,H), P_2) = E(M(C,H), M(K,Q))$$

By restricted commutativity between Mlt and Enc, $T_1 = T_2$

Parameters	
private key bytes	168
public key bytes	168
signature bytes	56

4. The Vec and Dup functions and Xifrat1-Kex.I

In this section, we present the Vec and the Dup function, discuss the properties needed for constructing key exchange from it, and present such construction.

4.1. Restricted Commutativity of Vec and Dup over themselves

The purpose of the Vec function is the same as that of the Blk function, except it works over a different domain. The Vec function works over 2 cryptograms that's made up of 7 63-bit slices similar to Enc and Mlt. The cryptogram is also 448-bit long with 441 effective bits. The construction of Vec is structurally similar to Blk.

Within the Vec function, each of the 63-bit slices are "hashed" in the Blk function, and applied sequentially twice interlaced with the other operand. An obvious flaw is that, if we can *individually* brutal-force the slices, then we can evaluate the key exchange maths, which is a fatal break. (This had been an oversight in the previous versions of this paper, which we fix now, by appending the Roman numeral ".I" to the name of the scheme.)

This is why, another layer is needed, which we call Dup. The purpose of Dup is, yet again, the same as Blk as well as Vec, but this time, the 7 slices are "hashed", requiring attacker to brutal force $7 \times 63 = 441$ bits. While this is a overkill for almost every scenario, we leave this as an overhead in case any powerful cryptanalytic attack is discovered.

Figure 4.1. The algorithm for the Vec function

- Input: $A=(A_0 A_1 \dots A_6)$, $B=(B_0 B_1 \dots B_6)$
- Output: $C=(C_0 C_1 \dots C_6)$

Steps:

- $C_0 = (A_0 A_1 \dots A_6) (B_0 B_1 \dots B_6) (A_0 A_1 \dots A_6) (B_0 B_1 \dots B_6)$
 - $C_1 = (A_1 A_2 \dots A_0) (B_1 B_2 \dots B_0) (A_1 A_2 \dots A_0) (B_1 B_2 \dots B_0)$
 - $C_2 = (A_2 A_3 \dots A_1) (B_2 B_3 \dots B_1) (A_2 A_3 \dots A_1) (B_2 B_3 \dots B_1)$
 - ...
 - $C_6 = (A_6 A_0 \dots A_5) (B_6 B_0 \dots B_5) (A_6 A_0 \dots A_5) (B_6 B_0 \dots B_5)$
-

Figure 4.2. The algorithm for the Dup function

- Input: $A=(A_0 A_1)$, $B=(B_0 B_1)$
- Output: $C=(C_0 C_1)$

Steps:

- $C_0 = (A_0 A_1) (B_0 B_1) (A_0 A_1) (B_0 B_1)$
- $C_1 = (A_1 A_0) (B_1 B_0) (A_1 A_0) (B_1 B_0)$

For ease of readability, $D(D(a,b),c)$ will be rewritten as $(a \cdot b \cdot c)$.

4.2. The Xifrat1-Kex.I KEM

As we've had the generalized restricted-commutativity property, we can construct a 9-variable key agreement scheme by laying out them in a square like this:

$$(abc)(def)(ghi) = (adg)(beh)(cfi)$$

For ease of visualized intuition, we lay them graphically:

```
// 9 variables //
a b c
d e f
g h i
```

The sum is identical regardless whether the matrix is evaluated row-first or column-first. Thus we take the middle column as the public key and the middle row as the ciphertext; b , h as "server-side" private key, and d , f as "client-side" secret share; The variables a , c , e , g , i are used as public information. The public key and the ciphertext are both in some kind of "sandwich" structure we mentioned in the previous section, which we believe makes recovering private keys impossible.

It is obvious at this point that the public information can be derived from a seed using some extendable output function (XOF), (prior art: [NewHope](#)). We instantiate such XOF with SHAKE-128. We take 448-bit in turn, interpret it as 7 64-bit unsigned integers in little-endian and clear each of their top bits, and generate 5 of these and assign them to a , c , e , g , i in order. We denote this XOF as $\text{Hx}_{[448-7] \times 2}(\text{seed})$.

Figure 4.3. Xifrat1-Kex.I Key Generation

1. Uniformly randomly generate choose a $seed$,
2. Generate a, c, e, g, i using $Hx_{[448-7] \times 2}(seed)$,
3. Uniformly randomly generate 2 cryptograms b, h ,
4. Compute $p = (b \cdot e \cdot h)$,
5. Return $pk = (seed, p)$ as the public key and $sk = (seed, b, h)$ as the private key.

Figure 4.4. Xifrat1-Kex.I Encapsulation

1. Expand $seed$ into a, c, e, g, i using $Hx_{[448-7] \times 2}(seed)$,
2. Uniformly randomly generate 2 cryptograms d, f ,
3. Compute $ss = (a \cdot d \cdot g) \cdot pk \cdot (c \cdot f \cdot i)$,
4. Compute $ct = (d \cdot e \cdot f)$,
5. Return ss as shared secret and ct as ciphertext.

Figure 4.5. Xifrat1-Kex.I Decapsulation

1. Return $ss = (a \cdot b \cdot c) \cdot ct \cdot (g \cdot h \cdot i)$ as shared secret.

Parameters	
private key bytes	232
public key bytes	120
ciphertext bytes	112

Annex A. References

- [AE] Derek Atkins, P. Gunnells; 2015; *Algebraic Eraser (TM) : A lightweight, efficient asymmetric key agreement protocol for use in no-power, low-power, and IoT devices*; <http://csrc.nist.gov/groups/ST/lwc-workshop2015/papers/session8-atkins-paper.pdf>
- [Kyber] Joppe W. Bos, L. Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, J. Schanck, P. Schwabe, and D. Stehlé; 2017-06 *CRYSTALS-Kyber: A CCA-Secure Module-Lattice-Based KEM* <https://pq-crystals.org/kyber> , <https://ia.cr/2017/634>
- [NewHope] Erdem Alkim and Léo Ducas and Thomas Pöppelmann and Peter Schwabe; 2015-11 *CRYSTALS-Kyber: A CCA-Secure Module-Lattice-Based KEM* <https://newhopecrypto.org> , <https://ia.cr/2015/1092>

- [Saber] Jan-Pieter D’Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren; 2017-06 *Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM* <https://ia.cr/2018/230>
- [SEC#1] *SEC 1: Elliptic Curve Crptography* <http://secg.org>
- [SIKE] David Jao, et al.; 2017 *SIKE – Supersingular Isogeny Key Encapsulation* <http://sike.org>
- [SQISign] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit and Benjamin Wesolowski; 2020-10 *SQISign: compact post-quantum signatures from quaternions and isogenies*; <https://ia.cr/2020/1240>
- [WalnutDSA] Iris Anshel, Derek Atkins, Dorian Goldfeld, and Paul E Gunnells; 2020-10 *WalnutDSA(TM): A Quantum-Resistant Digital Signature Algorithm*; <https://ia.cr/2017/058>
- [NN21] Daniel Nager, and Jianfang "Danny" Niu; 2020-10 *Xifrat - Compact Public-Key Cryptosystems based on Quasigroups*; <https://ia.cr/2021/444>