Quasigroups in Lightweight Cryptographic Hardware

Bernhard Birnbaum
bernhard.birnbaum@stud.tu-darmstadt.de

Joshua Liam Friedel
joshua.friedel@stud.tu-darmstadt.de

Hendrik Goeb
hendrik.goeb@stud.tu-darmstadt.de

Abstract

The increasing proliferation of low-resource devices, such as IoT sensors, RFID cards and embedded systems, demands cryptographic algorithms that are both secure but also resource-efficient. In this paper, we conceptually compare promising cryptographic design primitives, mainly focusing on quasigroup-based approaches. To draw conclusions about the usability and possible standardization in such constrained environments, we investigate the security, performance, complexity, efficiency and scalability of the algorithms.

On one hand, we take a look at quasigroup-based constructions for stream- and block-ciphers from [14] (to achieve confidentiality). On the other hand, we discuss approaches for block-based lightweight message authentication codes (MACs) from [7] (to achieve integrity and authentication). Both types of algorithms can also be combined to further optimize implementation for lightweight hardware (as quasigroup-based block-cipher MAC) which is why many quasigroup-based primitives have been proposed. For this reason, we have limited the selection of algorithms to cover all core security aspects that can be protected with cryptography.

Regarding stream ciphers, the most minimal but secure candidate is $\mathbf{ALG}_3$, and if basic computational capabilities are available one should use $\mathbf{ALG}_5$. For block ciphers, we found $\mathbf{ALG}_6$ to be simple, fast and secure, qualifying itself as basis for CBC MACs, whereas $\mathbf{ALG}_{10}$ turned out to be the best all-purpose construction in the comparison and $\mathbf{ALG}_{11}$ to be the most secure algorithm, as it has built-in error detection. Nevertheless, the most promising candidate for standardization purposes in constrained environment is probably $\mathbf{ALG}_9$; due to its inductive definition it can generate secure algorithms for any use case. Finally for the message authentication codes, OMAC seems to have the best tradeoff between efficiency, usability and complexity.

1 Introduction

The rapid rise of low-resource devices, e.g. IoT sensors, RFID cards, NFC tags, wearables or embedded systems in general, motivates the development of cryptographic algorithms that balance security, performance, complexity, efficiency and scalability. This research is essential to support the security of critical infrastructures operating under low-power conditions. Current challenges include ensuring strong cryptographic properties while minimizing energy consumption and silicon area, while resisting known attacks such as differential and linear cryptanalysis.

One promising option for such cryptographic algorithms are quasigroup-based encryption schemes [11]. Quasigroups (algebraic structures characterized by non-linearity and simple convertibility) provide an attractive foundation for designing lightweight cryptographic primitives and research has been carried out extensively. Some of the resulting modifications are based on T-quasigroups and systems of orthogonal n-ary groupoids. The inherent structural properties of such algebraic structures enable a compact implementation and resilience to certain cryptanalytic attacks, making them suitable for environments with tight energy and memory constraints.

In parallel, lightweight constructions like the One-Key CBC MAC can be used to forge efficient message authentication codes for arbitrary message lengths using minimal resources and only a single key, and serve as valuable benchmarks for assessing new designs.

Our initial selection of algorithms is motivated by the typical security goals that can be achieved with cryptography (confidentiality, integrity, and authenticity), so that limited hardware environments can also establish common security measures. This paper investigates and conceptually compares several quasigroup-based encryption algorithms from [14] and some MAC constructions from [7] by analyzing their feasibility on hardware platforms. The goal is to determine their potential use in real-world constrained environments and evaluate possible standardization in modern lightweight cryptographic frameworks regarding security, performance, complexity, efficiency and scalability.

More advanced algorithms, for instance [12], [11], [15] or [9], are outside the scope of this work, but still can give insights to further optimizations.

2 Preliminaries

In this section we briefly summarize the required preliminaries and mathematical notions. At first, section 2.1 defines general mathematical terminology and background on quasigroups including concepts related to block-based message authentication codes. Section 2.2 then introduces the basic cryptographic terms to cover the key concepts, including definition of ciphers and security notions.

2.1 Mathematical background

We briefly recall the algebraic structures relevant for quasigroup-based encryption [14, pp. 2-3]. First, a **group** is a set with an operation "$\cdot$" that combines two elements of the set into a third one. The operation "$\cdot$" has to satisfy associativity and every element of the set has to have an identity element and an inverse element. An n**-ary groupoid** is a non-empty set Q equipped with an n-ary operation $A : Q^n \to Q$ for $n \geq 2$ and is denoted by (Q,A). Building on groupoids, an n**-ary quasigroup** is a groupoid (Q,A), where the knowledge of any n elements uniquely specifies the remaining one. In any n-ary quasigroup, fixing argument $a \in Q$ of the operation A defines mappings $L_a(x) = a \cdot x$, $R_a(x) = x \cdot a$ and $x \cdot P_a(x) = a$, called (left, right and middle) **translations**, where $L_a(x)$ and $R_a(x)$ are bijective functions. Also, any n-ary quasigroup defines $((n+1)!-1)$ additional quasigroups, called **parastrophes** of (Q,A), where the arguments are permutated. For instance, when dealing with a binary groupoid (Q,A), it can be turned into a binary quasigroup by defining operations ${}^{(13)}A$ and ${}^{(23)}A$ (parastrophes) to create an algebra $(Q,A,{}^{(13)}A,{}^{(23)}A)$ (which is traditionally denoted as $(Q,\cdot,/,\backslash)$) such that the following identities are fulfilled:

1. $A({}^{(13)}A(x,y),y) = x$
2. ${}^{(13)}A(A(x,y),y) = x$
3. $A(x,{}^{(23)}A(x,y)) = y$
4. ${}^{(23)}A(x,A(x,y)) = y$

Moving on with **orthogonality** of quasigroups [14, pp. 8-9]: n-ary groupoids are called orthogonal, if for any fixed n-tuple $a_1,a_2,\ldots,a_n$ the system of equations $f_i(x_1,\ldots,x_n) = a_i : \forall i \in \{1,\ldots,n\}$ has a unique solution, which yields $(|Q|^n)!$ possible orthogonal permutations. Furthermore, an **automorphism** is a bijective map which has to preserve all operations and relations (isomorphism to itself) [14, p. 13]; it is used for orthogonality and efficient function inversion. This leads to the definition of T-quasigroups: A T**-quasigroup** is a special type quasigroup (Q,A) of the form $A(x,y) = \varphi x + \psi y + c$, where $(Q,+)$ is an abelian group, φ and ψ are automorphisms of $(Q,+)$ and $c \in Q$. The T-quasigroup also has **orthogonality**, if $(Q,\cdot) : x \cdot y = \alpha x + \beta y + c$ and $(Q,\circ) : x \circ y = \gamma x + \delta y + d$ both over $(Q,+)$ if the map $\alpha^{-1}\beta - \gamma^{-1}\delta$ is an automorphism of $(Q,+)$ [14, p. 17]. Finally we recall the **Klein group** which is defined as $\mathbb{Z}_2 \oplus \mathbb{Z}_2 = \{(0;0),(0;1),(1;0),(1;1)\}$ and its automorphism group [14, p. 16] $\mathrm{Aut}(\mathbb{Z}_2 \oplus \mathbb{Z}_2) = \begin{pmatrix}1&0\\0&1\end{pmatrix}, \begin{pmatrix}1&0\\1&1\end{pmatrix}, \begin{pmatrix}1&1\\0&1\end{pmatrix}, \begin{pmatrix}0&1\\1&0\end{pmatrix}, \begin{pmatrix}1&1\\1&0\end{pmatrix}, \begin{pmatrix}0&1\\1&1\end{pmatrix}$.

Next, we move on with mathematical preliminaries regarding lightweight message authentication codes: First, a **field** is a set F together with two binary operations on F called addition and multiplication, which require to satisfy the following field axioms: associativity, commutativity, distributivity, additive and multiplicative identity and inverse elements. One special kind of fields are **Galois fields** which contain finite number of elements; such finite fields are usually created by integers $\bmod q$ where q is prime and are denoted by $\mathrm{GF}(2^n)$. They are widely used in cryptography because of their specific properties: they are finite, efficient, secure and all arithmetic is closed. One element of such a field can be viewed in three equivalent ways [7, pp. 4]:

1. as an **abstract single point** in $\mathrm{GF}(2^n)$.
2. as an n-**bit string** $a_{n-1},\ldots,a_0 \in \{0,1\}^n$, where a_{n-1} is the most significant bit.
3. as a formal **polynomial** over $\mathrm{GF}(2)$: $a(u) = a_{n-1}u^{n-1} + a_{n-2}u^{n-2} + \cdots + a_1u + a_0$, where each coefficient is either 0 or 1.

This way, addition in $\mathrm{GF}(2^n)$ can be seen as bitwise $\oplus$ operation of two n-bit strings $a \oplus b$. To multiply two elements $a,b \in \mathrm{GF}(2^n)$, interpret them as polynomials and apply default polynomial multiplication, but with $\oplus$ as operation instead of integer addition, and finally reduce the result to n bits by modulo a fixed irreducible polynomial $f(u)$ of degree n (common choices are block sizes of $n = 256$, $n = 128$ or $n = 64$). A special and frequent case is multiplying/dividing by the field's generator u. This corresponds to shifting the bit string one position to the left/to the right, and if the most significant bit/least significant bit was 1, apply $\oplus$ to the result with a fixed constant R (to multiply)/R' (to divide) derived from $f(u)$ [7, p. 5].

2.2 Cryptographic background

In this section, we briefly recall some basic cryptographic notions that will be used throughout the paper. First of all, there are two main elementary cryptographical procedures [14, p. 2]:

1. **permutations** of plaintext symbols by some law,
2. **substitution** of plaintext symbols by some law.

For **bitstring operations**, let $a,b \in \{0,1\}^n$ be equal-length strings, then $a \oplus b$ denotes their bitwise $\oplus$ (XOR), $a \circ b$ denotes their concatenation and bitshift operations are written as $a \ll 1$ for left shift and $a \gg 1$ for right shift [7, p. 5]. It is also important to highlight the difference between stream and block ciphers [14, p. 2]: A **stream cipher** processes data sequentially bit by bit or byte by byte. This is achieved by generating a pseudo-random key stream derived from the secret key and then combining it with the plaintext using $\oplus$ operation. On the other hand a **block cipher** encrypts fixed-size

blocks of data. If the message exceeds the size of one block, it is split in multiple blocks, where the mode of operation specifies how to handle multiple blocks securely. Formally, a block cipher E is a function $E : \mathcal{K}_E \times \{0,1\}^n \to \{0,1\}^n$, where each $E(K,\cdot) = E_K(\cdot)$ is a permutation on $\{0,1\}^n$, $\mathcal{K}_E$ is the set of possible keys and n is the block length [7, p. 4]. The **Feistel scheme** is a symmetric structure to construct block ciphers by using a round function (a function which takes data block and a subkey as input) to encrypt data. As the name implies, the round function is applied each round to half of the data to be encrypted, while its output is XORed with the other half. Feistes networks provide interesting properties, especially that the entire operation is guaranteed to be invertible, even if the round function itself is not. One well-known mode of operation of block ciphers is **cipher block chaining** (CBC), where each plaintext block is processed by $\oplus$ with the previous ciphertext block prior encryption and the first block is initialized with an initialization vector. This chaining adds randomness and ensures that identical plaintext blocks encrypt differently depending on their position and previous blocks to prevent pattern leaks in encrypted data [7, p. 4]. The notation $\|a\|_n = \max\{1, \lceil |a|/n \rceil\}$ counts the number of n-bit blocks of any bitstring a and an empty string is handled as one block. To handle inputs that are shorter than a full block, a **padding function** is required: for a bitstring a with $|a| \leq n$, $\text{pad}_n(a)$ implements 10^i-padding as follows:

$$\text{pad}_n(a) = \begin{cases} a10^{n-|a|-1}, & |a| < n, \\ a, & |a| = n. \end{cases}$$

While cipher schemes ensure data confidentiality, **message authentication codes** (MACs) provide integrity and authenticity. Moving on to security properties, an encryption scheme is considered **resistant against statistical attacks** if there are no significant statistical patterns that allow derivation of any information about the plaintext [14, p. 4]. What is statistical resistance to encryption schemes is **resistance against birthday attacks** to MACs, whereby an attacker can't find a collision in fewer than $2^{n/2}$ steps, but beyond that point, security drops rapidly. If an attack requires up to 2^n operations, the algorithm is considered to be secure beyond the birthday paradox limit [7, p. 3].
Finally, we arrived at the standard notions of cryptographic security. An encryption scheme is said to be secure against chosen-plaintext attacks (**CPA**) if no attacker with access to an encryption oracle is able to distinguish the ciphertexts of any two chosen messages. A stronger notion is chosen-ciphertext attack (**CCA**) security, which is provided when an encryption scheme is secure if no attacker with access to an encryption and decryption oracle is able to distinguish the ciphertexts of any two messages, excluding the challenge's ciphertext [14].

3 Related Work/Literature Review

In this section, we go over a brief overview of recent works in lightweight cryptography. Besides our initial set of quasigroup-based approaches from [14] and lightweight block-based MACs from [7], research continues on other construction methods optimized for low-footprint, energy-efficient schemes for constrained devices. Therefore, it should be said that key efforts in the area of lightweight cryptography include firstly the *NIST Lightweight Cryptography Project*[1], which started in 2013 and recently standardized lightweight Ascon-based algorithms for constrained devices in 2023 [9, p. 3] [16]. Secondly, *ISO/IEC*-standards[2,3] proposed lightweight block ciphers such as PRESENT [4], CLEFIA, LEA and Grain-128a [11, pp. 1-2]. These algorithms implement alternative approaches and use different types of constructions rather than quasi-groups.
The following types of algorithms have been considered in research on lightweight cryptographic algorithms for low-resource environments, including quasigroup approaches and other well-known algorithms:

- **substitution-permutation networks** (SPNs):
  This is a very common design principle to construct block ciphers, e.g. AES is probably the most common SPN-based cipher. Although AES is too complex for lightweight applications (too much space and energy consumption), proposed ciphers like PRESENT [4] (standardized in ISO/IEC 29192-2:2019) use a reduced S-box size and simpler permutations to be more eglibible for constrained environments (very small silicon footprint and low gate count) [9, pp. 3, 7]. That said, SPN-based constructions can serve as valuable lightweight approaches, if they are proportioned appropriately. Other such candidates are PRINTcipher, Piccolo, LEA (standardized in ISO/IEC 29192-2:2019) and LED [6] (some weaknesses revealed in PRINTcipher, Piccolo and LED) [11, pp. 1-2].
- **feistel networks**:
  As Feistel constructions tend to have smaller circuits as not much parallelization is required, but on the other hand have to run more rounds to be secure, such schemes are less common for lightweight ciphers (well-known candidates are TWINE [11] and CLEFIA (standardized in ISO/IEC 29192-2:2019)).
- **quasigroup-based constructions**:
  Quasigroups can be used to construct both stream- and block-ciphers. Especially block-ciphers based on orthogonal systems of quasigroups have been shown to have very valuable and intersting properties for

1. NIST - Lightweight Cryptography, 2025: https://csrc.nist.gov/projects/lightweight-cryptography
2. ISO/IEC 29192-2 - block ciphers, 2019: https://www.iso.org/standard/78477.html
3. ISO/IEC 29192-8 - authenticated encryption, 2022: https://www.iso.org/standard/80114.html

resource-constrained environments [14]. The broad variety of quasigroup-based primitives proposed in the last years is remarkable: Edon-80 (stream cipher), Edon-R (hash function), INRU [15], BCWST, SEBQ [9, p. 3] (block ciphers), Q-S-box [12] (quasigroup-based S-boxes) and even MQQ (multivariate quadratic quasigroups), a quasigroup-based public-key block-cipher [9, p. 2]. This is the reason why existing lightweight ciphers like PRESENT [4], as an example for substitution-permutation networks, can also be enhanced by introducing quasigroup-based primitives (e.g. Q-S-box). It has been shown that such combinations can lead to 16% of area reduction compared to already parallelized implementations due to circuit reusage, as no additional registers are required, revealing the potential of optimization that quasigroups can offer for lightweight cryptographic algorithms [12, pp. 5, 7]; this is similar to how we can use quasigroup-based block-cipher as base of the **CBC MAC**.

- **sponge constructions**:
  Sponge constructions belong to the modern cryptographic methods and are very interesting for lightweight hardware applications due to their exceptionally high efficiency. As the sponge-based approach originates from hash constructions (e.g. GAGE), those algorithms can also be used to offer authenticated encryption (e.g. InGAGE or Ascon, which has been standardized in NIST SP 800-232). Ascon is efficient in both hardware and software, and balances performance, energy consumption and security by being more versatile than other block-ciphers, since it also supports authentication and hashing [9, p. 3].
- **other constructions**:
  There are also some design approaches which are not that common. One such candidate is Grain-128a (stream cipher), which is a lightweight authenticated encryption scheme, it is based on linear feedback shift registers and was standardized in ISO/IEC 29192-8:2022.

Completing this section now with a short discussion on why certain algorithms have gained acceptance in the scientific community, while others have not. Accepted schemes like Ascon, PRESENT, LEA, CLEFIA (and AES) benefit from extensive analysis, standardization and real-world adoption and can easily proven to be secure (SPN-based), whereas many quasigroup-based algorithms still remain theoretical, with limited and practical deployment.

4 Lightweight cryptographic algorithms

This chapter introduces the algorithms for comparison. As we initially started with [14] and [7] to cover both ciphers and MACs, this sections is structured as follows: While section 4.1 covers quasi-group-based encryption algorithms, section 4.2 deals with block-based message authentication codes.

Quasigroups play a vital role in the construction of lightweight cryptographic schemes and are therefore used in a wide variety of primitives, as they yield the same permutations and substitutions as common ciphers do, but at a fraction of the cost [14, p. 2]. Obviously we can use them to develop stream- and block-cipher schemes [14], which we can use in a next step to construct quasigroup-based MACs [7]; however also quasigroup-based S-Boxes [12] and even more complex algorithms like PRESENT and TWINE [11], INRU [15] and SEBQ [9] were proposed, but are not covered in this paper. Nevertheless there are also quasigroup-based analogues of existing algorithms which were found to have a 30-45% reduction of silicon area, but also only 25% of the data throughput of the original cipher [12, p. 6]. Since performance of algorithms in lightweight environments often is no critical property, it could be worth integrating these approaches into the comparison in future work (see section 6.2).

4.1 Quasigroup-based cryptosystems

The relations of the considered quasigroup-based ciphering schemes can be seen in Figure 1. While the first five algorithms **ALG**$_1$ to **ALG**$_5$ are stream ciphers (which are in some cases mandatory when buffering is limited or when characters must be individually processed [14, p. 2]), **ALG**$_6$ to **ALG**$_{11}$ realize block-based approaches (making them usable as primitive for block-based MACs like in section 4.2).

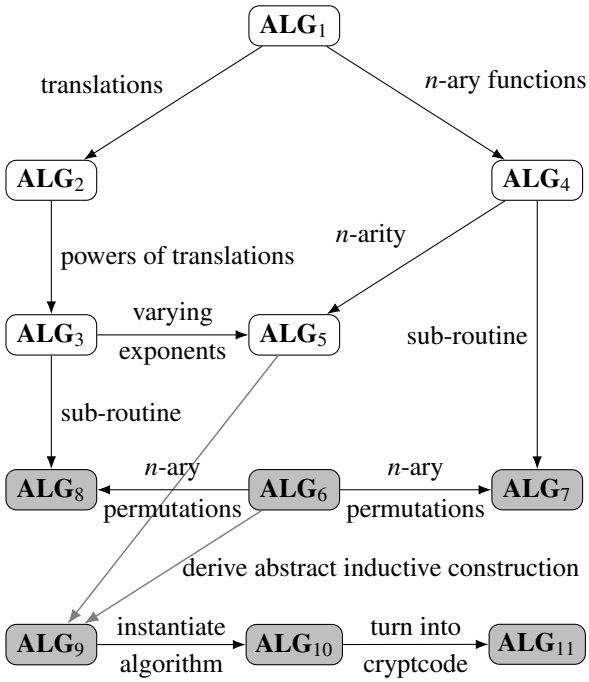

Figure 1: Relations of the algorithms from [14], whereby the highlighted nodes represent block-based approaches

- **ALG**$_1$ [14, pp. 3-4]:
  This basic algorithm implements the idea of using a binary quasigroup (Q,A) to construct a simple stream cipher. It is built on a central property of quasigroups: given equation $A(x,y)=z$, knowing any two of x, y or z is sufficient to uniquely determine the third value. Therefore, for any binary quasigroup and its corresponding $^{(23)}$-parastrophe it holds that $x = {}^{(23)}A(y,A(y,x))$. Every plaintext symbol is encrypted using a quasigroup operation involving the previous ciphertext symbol, starting from a fixed "leader"-element $l \in Q$ and proceeding sequentially: $v_1 = A(l,u_1), v_i = A(v_{i-1},u_i)$. Decryption uses the $^{(23)}$-parastrophe of A, which serves as inverse operation to recover the plaintext.
- **ALG**$_2$ [14, pp. 4-5]:
  The second algorithm rewrites **ALG**$_1$ using the concept of translations, which is used to express the encryption as sequential application of (bijective) functions instead of group operations. For this reason, encryption can be denoted as follows: $v_1 = L_l(u_1), v_i = L_{v_{i-1}}u_i$.
- **ALG**$_3$ [14, pp. 5-6]:
  A third revision of **ALG**$_2$ adds a generalization by allowing powers of the translations, so L_a^k is used instead of L_a, with varying exponents at each step. This way, complexity and variability is being added to the cipher, enhancing the resistance to attacks, as the security is not dependend of the "leader"-element only anymore: $v_1 = L_l^{c_1}(u_1), v_i = L_{v_{i-1}}^{c_i}u_i$.
- **ALG**$_4$ [14, pp. 6-7]:
  Whereas **ALG**$_1$ is based on *binary* quasigroups, **ALG**$_4$ generalizes this idea for n-ary analogues. By definition, for an n-ary quasigroup operation f the knowledge of any n elements uniquely determines the remaining one, whereas it holds that $x = {}^{(n,n+1)}f(\dots,f(\dots,x))$. Therefore, to encrypt the first $n-1$ characters, $(n-1)^2$ fixed "leader"-elements are required: $v_1 = f(l_1 \dots l_{n-1},u_1), v_{n-1} = f(l_{n^2-3n+3} \dots l_{(n-1)^2},u_{n-1})$. Subsequently, every following plaintext symbol is encrypted using the previous $n-1$ ciphertext symbols: $v_n = f(v_1 \dots v_{n-1},u_n), v_{n+1} = f(v_2 \dots v_n,u_{n+1}),\dots$.
  The decryption works the same way by delegating the ciphertext to the $^{(n,n+1)}$-parastrophe of f. In comparison to **ALG**$_1$, **ALG**$_4$ introduces $(n-1)^2$ "leader"-elements and requires to maintain $n-1$ ciphertext symbols for each step, which increases the computational complexity and memory usage.
- **ALG**$_5$ [14, pp. 7-8]:
  Essentially, **ALG**$_5$ is a combination of the concepts behind **ALG**$_3$ and **ALG**$_4$. Whereas in **ALG**$_3$ it is proposed to use powers of translations with variable exponents to increase security and **ALG**$_4$ relies on an n-ary quasigroup structure instead of just binary operations, **ALG**$_5$ makes use of both ideas. Accordingly, the encryption of the first $n-1$ symbols again require $(n-1)^2$ "leader"-elements plus varying exponents for each symbol and is carried out as follows: $v_1 = T^{c_1}(l_1 \dots l_{n-1},u_1), v_{n-1} = T^{c_{n-1}}(l_{n^2-3n+3} \dots l_{(n-1)^2},u_{n-1})$. For any following character, the previous $n-1$ ciphertext symbols are used: $v_n = T^{c_n}(v_1 \dots v_{n-1},u_n), v_{n+1} = T^{c_{n+1}}(v_2 \dots v_n,u_{n+1}),\dots$.
- **ALG**$_6$ [14, pp. 9-10]:
  Instead of just relying on a single n-ary quasigroup, **ALG**$_6$ makes use of orthogonality of n-ary operations to achieve a more uniform distribution of elements in the base set, which enhances protection against statistical cryptanalysis attacks. A system of n n-ary orthogonal operations $(A,f_i) : i \in \{1,\dots,n\}$ defines a permutation F on the set A^n which can be used for encryption, as the corresponding system of equations has a unique solution. **ALG**$_6$ applies permutations to a block of n characters and uses a sliding window mechanism combined with the idea of powers (similar to translations in **ALG**$_3$), therefore the encryption of the first blocks can be denoted by $y_1^n = F^l(x_1^n), z_1^n = F^s(y_2,y_3,\dots,y_n,x_{n+1}),\dots$; use of Feistel scheme is also possible.
- **ALG**$_7$ [14, pp. 10-11]:
  ALG$_7$ makes use of **ALG**$_4$ (n-arity) and **ALG**$_6$ (block permutation): in a first step, plaintext is divided into n-tuples, which are then processed with n-ary permutations F^s and F^l similar to **ALG**$_6$. Subsequently **ALG**$_4$ is applied to every tuple (G), followed by another n-ary permutation; encryption per block can be denoted by $t_1^n = F^s(G(F^l(x_1^n)))$.
  ALG$_8$ [14, pp. 11-12]:
  In comparison to **ALG**$_7$, **ALG**$_8$ combines **ALG**$_3$ (powers of translations) and **ALG**$_6$ (block permutation). Plaintext is processed pairwise in binary tuples, where **ALG**$_3$ (T) is applied for each plaintext symbol, starting from the first block: $(v_1,v_2) = F_1^a({}_{g_1}T_{l_1}^b(u_1),{}_{g_2}T_{l_2}^c(u_2)), (v_3,v_4) = F_2^d({}_{g_3}T_{v_1}^e(u_3),{}_{g_4}T_{v_2}^f(u_4)), (v_5,v_6) = F_3^g({}_{g_5}T_{v_3}^h(u_5),{}_{g_6}T_{v_4}^i(u_6))$.
- **ALG**$_9$ [14, pp. 15-16]:
  ALG$_9$ provides an abstract method to construct cryptographical procedures based on the previously discussed functions. For one thing, function $F(x_1^n)$ denotes a permutation of the set Q^n based on n orthogonal groupoids. However, function $g(\overline{a}^{n-1},x)$ is n-ary quasigroup operation, which makes $g(\overline{a}^{n-2},x,y) = b$ hard by definition. It is proposed to define cryptographical terms inductively: individual constants, individual variables, g (if it's n-ary quasigroup functional and its arguments are individual constants or terms) and F (if it's a permutation on Q^n constructed by n orthogonal n-ary groupoids and its arguments are cryprographical terms) are cryptographical terms. This way, **ALG**$_9$ acts as a framework to construct related algorithms with guaranteed cryptographic properties.

- **ALG**$_{10}$ [14, pp. 15-16]:
  ALG$_{10}$ uses definition of **ALG**$_9$ to construct the following cipher, which can be seen as generalization of **ALG**$_8$: given n-ary permutation F, n procedures G_j (arity may vary) and corresponding "leader"-elements. The i-th step of the algorithm can be denoted by ${}_iF^k(G_1(y_1^m,x_i),\ldots,G_n(y_1^r,x_i)) = {}_iy_1^n$.
- **ALG**$_{11}$ [14, pp. 16-19]:
  Last but not least, **ALG**$_{11}$ is a combined cryptcode using results from **ALG**$_9$ and **ALG**$_{10}$ plus some properties of T-quasigroups for introducing an error-detecting code. For construction, the Klein group $(\mathbb{Z}_2 \oplus \mathbb{Z}_2)$ is used due to its easy structure but non-trivial automorphisms, which are again necessary to construct orthogonal T-quasigroups. T-quasigroup definition includes two automorphisms (corresponding to Klein group pairs), but also a fixed element of the base set can be used for error-detection ($z = \varphi_5 x + \varphi_6 y$), which means two plaintext values correspond to three values in the cryptcode. This way, a triple of orthogonal operations $K_1(x,y,z), K_2(x,y,z), K_3(x,y,z)$ defines a permutation on set Q^3 denoted by K. Together with another permutation defined by a system of three ternary orthogonal groupoids (denoted by M), the following cryptographic term (see **ALG**$_9$) is composed: $H(x,y,z) = M^k(K^l(x,y,z)) : k,l \in \mathbb{Z}$. Encryption is carried out as follows: take next pair of information symbols $a,b \in (\mathbb{Z}_2 \oplus \mathbb{Z}_2)$, calculate $z = \varphi_5 x + \varphi_6 y$ using automorphisms, apply $H(a,b,c)$, take next pair and repeat.

4.2 Derivations of CBC MAC

The connections between the considered message authentication code constructions are depicted in Figure 2.

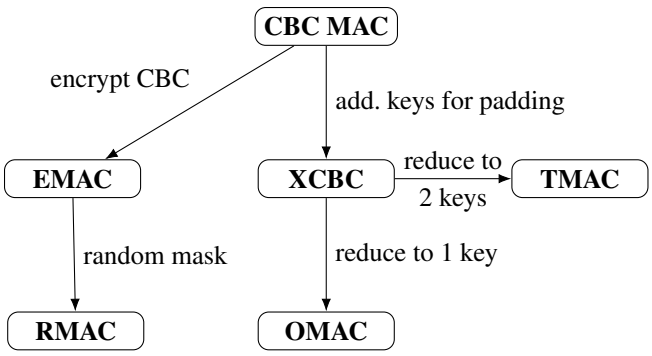

Figure 2: Relations of the MACs from [7]

- **CBC MAC** [7, pp. 1, 4]:
  The **CBC MAC** is a popular construction based on block ciphers, but its basic form is secure only for fixed-length messages. Several variants have been developed to handle variable lengths efficiently. To construct a message authentication code $\mathrm{CBC}_K(M)$ from a block cipher E for message $M = M[1] \circ M[2] \circ \cdots \circ M[m]$ with $n = |M[1]| = |M[2]| = \cdots = |M[m]|$, a very common and well-known approach is defined as $Y[m]$, where $Y[i] = E_K(M[i] \oplus Y[i-1])$ and $Y[0] = 0^n$. This **CBC MAC** has the well known problem that they need a fixed message length of mn bits to be secure.
- **EMAC** [7, p. 1]:
  The encrypted MAC (**EMAC**) is obtained by encrypting the **CBC MAC** value again with a second key: $\mathrm{EMAC}_{K_1,K_2}(M) = E_{K_2}(\mathrm{CBC}_{K_1}(M))$. This construction is secure if the message length is a positive multiple of n, but it requires two key schedulings.
- **XCBC** [7, pp. 1-2]:
  To generalize this idea for arbitrary message lengths, **XCBC** uses three keys to encrypt messages with variable length (see Figure 3). The first key K_1 is used to encrypt each block. The second key K_2 will be used when the message fits with the block size ($|M| = mn$, no padding required) and is applied by a XOR gate to the last block prior encryption. In case the message length is not divisible by the block size, padding of $i = n-1-|M| \mod n$ bits must be appended to fill the last block and K_3 is applied in the same manner as K_2. Although **XCBC** uses three keys ($k+2n$ bits in total), it requires only one key scheduling of the underlying block cipher E. **XCBC** is similar to **CBC MAC**, but with XORing different keys on the last block depending on whether padding is used.
- **TMAC** [7, pp. 2-3]:
  The **TMAC** construction is derived from **XCBC** and was introduced to reduce the key size to $k+n$ bits in total. **TMAC** only requires two keys: a block-cipher key and an n-bit key (both keys can be obtained using only one key scheduling). To obtain **TMAC** from **XCBC**, set $K_2 \to K_2 \cdot u$ and $K_3 \to K_2$, where u is some non-zero constant and "$\cdot$" denotes multiplication in $\mathrm{GF}(2^n)$; see section 2.1 for details.
- **RMAC** [7, p. 3]:
  Another proposed algorithm is random MAC (**RMAC**), an extension of **EMAC** using an n-bit random string, which acts as a mask on the key and is part of the MAC tag: $\mathrm{RMAC}_{K_1,K_2}(M) = (E_{K_2 \oplus R}(\mathrm{CBC}_{K_1}(M)), R)$. It is interesting to be considered, since it has been shown that the security of **RMAC** is beyond the birthday paradox limit [7, p. 3].
- **OMAC** [7, pp. 2-3, 5-6]:
  Similar to **TMAC**, the One-Key CBC MAC (**OMAC**) is a construction that aims to reduce the key size even further to only a single k-bit key K of the block cipher E in order to adapt **XCBC** for efficient implementation in hardware-constrained environments. To shrink the key size further than in **TMAC** while preserving its properties, **OMAC** generates L from K rather than storing it

as part of the key (see Fig. 3). This makes the security proof more challenging, yet **OMAC** remains as secure as **XCBC** and works for arbitrary message lengths while requiring only one key scheduling. In **OMAC** and related MAC constructions, certain computations are carried out in the finite field GF(2^n). The representation as GF(2^n) allows efficient key-dependent operations in **OMAC**. Multiplication or division by u requires at most one bit shift and one conditional XOR, making these operations very fast in practice, see section 2.1 for details. **OMAC** is a generic name for two variants of the algorithm, OMAC1 and OMAC2, which both work similar to **XCBC**: If message length $|M|$ is a multiple of n, **OMAC** works like **CBC MAC** (but $\oplus\, L \cdot u$ as key before the last encryption), otherwise the message is padded with $i = n - 1 - |M| \mod n$ bits.

- OMAC1: For OMAC1 (K_2, K_3) of **XCBC** is replaced by $(L \cdot u, L \cdot u^2)$.
- OMAC2: OMAC2 replaces (K_2, K_3) with $(L \cdot u, L \cdot u^{-1})$ with u being a non-zero constant in GF(2^n) and L given by $L = E_{K_1}(0^n)$.

The key replacements $(L \cdot u)$, $L \cdot u^{-1}$ and $L \cdot u^2$) are efficiently calculated with a shift and an conditional XOR with L, L or $(L \cdot u)$.

5 Comparison of algorithms

In this section, we compare the presented quasi-group-based encryption algorithms and MAC constructions. First, in section 5.1, we establish the methodology for the comparison. In section 5.2, we then apply it to the quasi-group-based algorithms and, in section 5.3, to the MAC algorithms. Finally in section 5.4 we evaluate the results.

5.1 Methodology

The comparative methodology applied in this work is designed to not only establish theoretical distinctions between the different quasigroup-based cryptographic schemes but also to reflect their feasibility in a real world environment. This is particularly relevant for modern lightweight cryptographic algorithms, which are meant to be deployed in constrained environments like embedded systems, IoT devices and RFID tags. In these environments the challenge is the limitation of processing capacity, memory and available energy. Accounting for these constraints, the evaluation proceeds through a structured set of categories that capture both cryptographic robustness and implementation feasibility.
To efficiently compare the selected quasigroup-based cryptographic schemes and the block-based MACs in terms of their cryptographic strength, performance and resource consumption, five different categories are introduced in order to derive conclusions about the potential standardization in modern lightweight cryptographic frameworks. Therefore, a specific feature was selected for each category of the comparison to enable later assessment:

1. **Security**: instantiated by cryptanalytic resistance
   $\rightarrow$ the higher, the better
2. **Performance**: instantiated by throughput/cipher speed
   $\rightarrow$ the higher, the better
3. **Complexity**: instantiated by gate count/silicon area
   $\rightarrow$ the lower, the better
4. **Efficiency**: instantiated by energy consumption
   $\rightarrow$ the lower, the better
5. **Scalability**: instantiated by parallelizability
   $\rightarrow$ the higher, the better

While considering security by cryptanalytic resistance, the performance, complexity, efficiency and scalability give insights about the algorithm's feasability in lightweight implementations. Along the comparison, we summarize each category by assigning a qualitative label starting from *very low*, *low*, *moderate*, *high* to *very high* (see Table 1 and Table 2). Those qualitative labels should be considered to be relative to lightweight quasigroup implementations (intra-quasigroup comparison); as we did not do any measurements using real implementations, an inter-cipher comparison with default implementations like AES is beyond the scope of this work (see section 6.2), since there are magnitudes between the performance of quasigroup-based ciphers and AES [11, p. 4].

5.2 Quasigroup-based cryptosystems

In this paragraph we present the results of the comparison of the quasigroup-based algorithms discussed in section 4.1, which are also summarized in Table 1.

1. **Security** (cryptanalytic resistance):
   The first two proposed algorithms using the properties of quasigroups are $\mathbf{ALG}_1$ and $\mathbf{ALG}_2$, which implement basic stream ciphers. However, in [17] it has been proven that this cipher is not CPA-secure and therefore also not CCA-secure, plus it is claimed that it is vulnerable to special kind of statistical attacks [14, p. 4]. Since both algorithms carry out exactly the same calculations (just mathematically rewritten), their security properties are identical. As $\mathbf{ALG}_3$ introduces powers to the translations, the construction is CPA- and CCA-secure under the condition that the exponents vary from step to step [14, p. 6]. Another possible generalization of $\mathbf{ALG}_1$ is $\mathbf{ALG}_4$, working on n-ary instead of binary group structures. Therefore it required more than one leader element, but unfortunately this algorithm still does not achieve CPA-/CCA-security [14, p. 5]. To fix this issue, $\mathbf{ALG}_5$ applies $\mathbf{ALG}_4$'s idea of generalization to n-ary groups to $\mathbf{ALG}_3$. As the security of $\mathbf{ALG}_3$ was already proven in [14, p. 6], we know that $\mathbf{ALG}_5$ is also secure if the exponents vary on each step. Using orthogonal systems of quasigroups as in $\mathbf{ALG}_6$ has the advantage

Algorithms	Comparison Metrics				
	Security	**Performance**	**Complexity**	**Efficiency**	**Scalability**
$\mathbf{ALG}_1$/$\mathbf{ALG}_2$ (stream ciphers based on binary quasigroup translations)	*low* resistance; not CCA-/CPA-secure [17]; vulnerable to statistical attacks [14, p. 4]	*very high* throughput;	*very low* gate count;	*very low* energy consumpt.;	*very low* parallelizability; simple table lookups and inherently sequential [14, p. 4]; constant time and gate count [14, pp. 4-5]; very efficient but not parallelizable (possible permutations scale linearly) due to stream character [14, pp. 3-5, 19]
$\mathbf{ALG}_3$ (stream cipher based on $\mathbf{ALG}_2$ plus powers of translations)	*moderate* resistance; CCA-/CPA-secure if exponents vary from step to step [14, p. 6]	*very high* throughput;	*low* gate count;	*very low* energy consumpt.;	*very low* parallelizability; increased computation, gate count and energy consumption due to exponentiation of translations [14, pp. 5-6]; varying exponents make handling of larger message more complex [14, pp. 6, 19]; yields up to $n!$ permutations [14, p. 19]
$\mathbf{ALG}_4$ (stream cipher based on n-ary analogue of $\mathbf{ALG}_1$/$\mathbf{ALG}_2$)	*low* resistance; not CCA-/CPA-secure [5, pp. 4-5] [14, p. 5]	*high* throughput;	*moderate* gate count;	*low* energy consumption;	*very low* parallelizability; n-arity increases complexity per step (more than one leader element required) [14, pp. 6-7]; yields up to $n!$ permutations [14, p. 19]
$\mathbf{ALG}_5$ (stream cipher based on n-ary analogue of $\mathbf{ALG}_3$)	*moderate* resistance; CCA-/CPA-secure if exponents vary from step to step [14, p. 6]	*high* throughput;	*moderate* gate count;	*low* energy consumption;	*very low* parallelizability; most complex stream cipher in the comparison, combines n-arity with translations [14, p. 8]; still efficient cipher, obtains $n!$ permutations of base set with size n [14, p. 19]
$\mathbf{ALG}_6$ (block cipher based on orthogonal systems of n-ary quasigroups)	*high* resistance; sufficiently safe against CCAs/CPAs and resistant to statistical attacks [5, p. 7] [14, p. 10]	*low* throughput;	*moderate* gate count;	*moderate* energy consumpt.;	*high* parallelizability; implements permutations based on orthogonal n-ary groupoids, turning it into block cipher [14, pp. 8-10]; powerful algorithms, allows up to $(q^n)!$ permutations [14, pp. 10, 19]; block-based character enables parallelization, but throughput is limited and circuit complexity is higher [14, p. 2]
$\mathbf{ALG}_7$ (block cipher based on combination of $\mathbf{ALG}_4$ and $\mathbf{ALG}_6$)	*high* resistance; increased safety against CPAs/CCAs compared to $\mathbf{ALG}_6$ [14, p. 11]	*low* throughput;	*moderate* gate count;	*high* energy consumption;	*high* parallelizability; as its a combined algorithm it has increased computational and memory requirements [14, p. 11]; less throughput as $\mathbf{ALG}_6$ is applied two times per block with different shift [14, p. 11]; block-wise operations are parallelizable, but application of $\mathbf{ALG}_4$ on a single block is not [14, p. 11]
$\mathbf{ALG}_8$ (block cipher based on combination of $\mathbf{ALG}_3$ and $\mathbf{ALG}_6$)	*high* resistance; CCA-/CPA-secure if exponents vary from step to step [14, p. 12]	*moderate* throughput;	*moderate* gate count;	*moderate* energy consumpt.;	*moderate* parallelizability; throughput is slightly higher as $\mathbf{ALG}_6$ is applied only one time [14, p. 11-12]; gate count and energy consumption is similar to $\mathbf{ALG}_7$ [14, p. 11-12]; depending on the implementation, this algorithm allows to obtain almost "natural" stream cipher, making it harder to parallelize [14, p. 11-12]
$\mathbf{ALG}_9$/$\mathbf{ALG}_{10}$ (abstract stream-based block cipher)	*high* resistance; CCA-/CPA-secure [14, p. 16]	*moderate* throughput;	*high* gate count;	*high* energy consumpt.;	*high* parallelizability; throughput, gate count and energy consumption depend on the specific implementation of the algorithm (uses n-ary quasigroup functional g on (Q,g) and permutation F of set Q^n), but is generally comparable to $\mathbf{ALG}_8$ [14, pp. 15-16]; it is parallelizable on block level and therefore highly scalable [14, p. 16];
$\mathbf{ALG}_{11}$ (stream-based block cipher as cryptcode)	*very high* resistance; CCA-/CPA-secure; cryptcode with check symbols [14, p. 16]	*very low* throughput;	*very high* gate count;	*very high* energy consumpt.;	*moderate* parallelizability; introduction of cryptcode significantly increases complexity and energy consumption, resulting in reduced throughput (ciphertext size increases by 50%) [14, p. 18-19]; parallelization is bad as the cipher works on pairwise elements to integrate the check symbols (almost stream cipher) [14, pp. 16-19]

Table 1: Conceptual comparison of quasigroup-based cryptographic algorithms

that there are $(q^n)!$ possibilities for constructing such systems (where q is the order of the set and n is the arity of the group), which greatly increases randomness. This leads to a more even distribution of the elements of the base set, which is why such systems are sufficiently safe against CPAs/CCAs and therefore also more preferable in protection against statistical cryptanalytic attacks [14, p. 10]. To make chosen cipher-/plainttext attacks even more complicated, $\mathbf{ALG}_7$ combines the properties of $\mathbf{ALG}_4$ and $\mathbf{ALG}_6$ by applying both algorithms simultaneously [14, p. 11]. As $\mathbf{ALG}_4$ is not CPA-/CCA-secure, $\mathbf{ALG}_6$ is merged with $\mathbf{ALG}_3$ instead to construct $\mathbf{ALG}_8$, which is proven to be secure if exponents vary from step to step [14, p. 12]. The following algorithms $\mathbf{ALG}_9$, $\mathbf{ALG}_{10}$ and $\mathbf{ALG}_{11}$ are all CPA-/CCA-secure, as they are based on an inductive definition ($\mathbf{ALG}_9$), where elements can be permutated and/or changed by proven secure functionals [14, p. 16].

2. **Performance** (throughput/cipher speed):
   First of all, it is clear that stream ciphers are generally faster than block ciphers, which implies that $\mathbf{ALG}_1$ to $\mathbf{ALG}_5$ are more performant and have higher throughput [14, p. 2]. Therefore $\mathbf{ALG}_1$ and $\mathbf{ALG}_2$ being just table lookups and not very heavy computational-wise, both algorithms are very performant [14, p. 4]. In $\mathbf{ALG}_3$ there is a slightly lower throughput than in $\mathbf{ALG}_1$ due to simple quasigroup translations, with increase computation from exponentiation of translations [14, p. 5-6]. This does reflect that it's still a lightweight stream cipher like $\mathbf{ALG}_2$, but the added powers/exponents introduce minimal extra computation per symbol. Moving on with sequential stream cipher $\mathbf{ALG}_4$, n-ary operations increase per-symbol computation even further compared to binary cases $\mathbf{ALG}_1$/$\mathbf{ALG}_2$ [14, pp. 6-7]. For the last stream cipher $\mathbf{ALG}_5$, due to n-arity combined with variable powers of translations, per-symbol complexity is the highest among the stream ciphers. Nevertheless $\mathbf{ALG}_5$ is still a quite fast encryption scheme [14, p. 8]. Moving on with the quasigroup-based block ciphers, $\mathbf{ALG}_6$ implements permutations using n-ary groupoids; since its a block-based construction, it is clear that performance is worse than $\mathbf{ALG}_5$ [14, p. 2]. $\mathbf{ALG}_7$ combines n-ary per-

mutations of $\mathbf{ALG}_6$ with $\mathbf{ALG}_4$, resulting in increased security but decreased throughput, as block permutation $\mathbf{ALG}_6$ is applied two times per block [14, p. 11]. In comparison, $\mathbf{ALG}_8$ combines $\mathbf{ALG}_6$ with $\mathbf{ALG}_3$, turning the algorithm into an almost stream cipher (processing pairs of elements), and as $\mathbf{ALG}_6$ is only applied once per block, throughput in total is higher than $\mathbf{ALG}_7$ [14, pp. 11-12]. $\mathbf{ALG}_9$/$\mathbf{ALG}_{10}$ are pretty close to $\mathbf{ALG}_8$ and just obtain generalizations, which is why throughput depends on the specific implementation [14, p. 16], but for $\mathbf{ALG}_{10}$ we know throughput is acceptable if binary case is instantiated (see $\mathbf{ALG}_8$). Finally, regarding throughput, $\mathbf{ALG}_{11}$ appears to be the slowest algorithm in the comparison, which is caused mainly by the introduced check symbols and their computation. Additionally to the increased overhead, ciphertext length will be 50% higher compared to the plaintext, as one check symbol is calculated pairwise for every two plaintext symbols [14, pp. 18-19].

3. **Complexity** (gate count/silicon area):
   Stream ciphers appear to have less complex hardware circuitry, which implies that $\mathbf{ALG}_1$ to $\mathbf{ALG}_5$ do not require many gates and take relatively low silicon area compared to block-based approaches as circuitry can be heavily reused [14, p. 2]. For $\mathbf{ALG}_1$/$\mathbf{ALG}_2$, the complexity in terms of gate count is constant per processed symbol, because each step performs one binary quasigroup operation which can be implemented as a table lookup or fixed logic circuit [14, p. 4-5]. The complexity of $\mathbf{ALG}_3$ is slightly higher than $\mathbf{ALG}_2$ because it uses powers of translations instead of plain translations, which adds extra permutation steps. As exponents vary on each step to remain secure, handling of larger messages is more complex [14, pp. 6, 19]. Since $\mathbf{ALG}_4$ uses an n-ary quasigroup instead of a binary one, the complexity per step increases with n due to more inputs being processed in each operation. Gate count depends on the implementation of the n-ary operation but is obviously generally higher than $\mathbf{ALG}_1$/$\mathbf{ALG}_2$, also additional leader elements are required. Overall, complexity still scales linearly with the number of symbols, but with a larger constant factor [14, pp. 6-7]. $\mathbf{ALG}_5$ combines n-arity with varying exponents in the powers of translations, so each encryption step requires computing the n-ary operation plus the application of translation powers, which increases the gate count compared to $\mathbf{ALG}_4$ slightly [14, p. 8]. Continuing with block ciphers, $\mathbf{ALG}_6$ uses an orthogonal system of n-ary quasigroups. This increases the computational cost compared to single n-ary transformations, with complexity growing linearly with block size and the number of quasigroups, which results in higher gate count due to multiple n-ary operations per block [14, pp. 10-11]. As $\mathbf{ALG}_7$ is a combination of the orthogonal operations from $\mathbf{ALG}_6$ and n-arity from $\mathbf{ALG}_4$, its gate count is roughly the sum of those, although $\mathbf{ALG}_6$ is called two times, the circuit can be reused. It's obvious that combined algorithms are significantly heavier than either alone [14, p. 11]. Similar to $\mathbf{ALG}_7$, $\mathbf{ALG}_8$ is also a combination of orthogonal operations from $\mathbf{ALG}_6$ and quasigroup translations from $\mathbf{ALG}_3$. Since it is designed as an almost stream cipher, its gate count should be slightly lower than of $\mathbf{ALG}_7$ [14, pp. 11-12, 19]. For $\mathbf{ALG}_9$/$\mathbf{ALG}_{10}$, we estimate higher gate count than earlier algorithms since they are not limited to almost stream cipher character, but $\mathbf{ALG}_9$ can be instantiated as full block cipher like $\mathbf{ALG}_{10}$. The exact gate count is hard to estimate, as the arity of the groups can vary along the ciphering [14, pp. 15-16]. Finally, $\mathbf{ALG}_{11}$'s gate count is dominated by computing check symbols, which is why it's the algorithm with the highest gate count in the comparison. It is otherwise comparable to the constructions of $\mathbf{ALG}_8$ (also almost stream cipher), but its output length is 150% of the input length [14, pp. 16-19].
4. **Efficiency** (energy consumption):
   In terms of efficiency, $\mathbf{ALG}_1$/$\mathbf{ALG}_2$ are very lightweight since they rely only on binary quasigroup translations and simple iterations. Their energy consumption is low, making them efficient for constrained devices [14, pp. 3-5]. Moving on to $\mathbf{ALG}_3$, we also estimate very low energy consumption overall, but slightly higher than $\mathbf{ALG}_1$/$\mathbf{ALG}_2$ due to computing powered translations each step. Still being sequential per symbol is the reason why no intra-message energy amortization via parallelism [14, p. 5-6]. Further with $\mathbf{ALG}_4$, which does have low energy demand because of higher per-symbol cost (n-arity) than $\mathbf{ALG}_1$-$\mathbf{ALG}_3$, but still efficient for small n. For $\mathbf{ALG}_5$, energy consumption is slightly above $\mathbf{ALG}_4$'s demand, since it combines n-ary operations with powered translations. It is still efficient for moderate n, but more costly per symbol than $\mathbf{ALG}_1$-$\mathbf{ALG}_4$ [14, pp. 8, 19]. Continuously $\mathbf{ALG}_6$ has moderate energy consumption: multiple n-ary orthogonal permutations (often across several rounds or Feistel steps) raise per-block cost and energy scales with the number of operations and arity, making it less efficient than the stream variants, though block-level parallelism can partially amortize it [14, pp. 10-11, 19]. Proceeding now to $\mathbf{ALG}_7$, which does have high energy consumption, as it combines n-ary stream operations of $\mathbf{ALG}_4$ with the multi-operation block structure of $\mathbf{ALG}_6$, so each encryption involves many n-ary transformations and orthogonal system computations [14, p. 11]. Continuing with $\mathbf{ALG}_8$, which has only small arity due to its almost stream cipher design, therefore consumes less energy than $\mathbf{ALG}_7$ [14, pp. 11-12, 19]. Moving on to $\mathbf{ALG}_9$/$\mathbf{ALG}_{10}$, being abstract and hybrid, energy consumption depends on specific implementation, but generally higher than $\mathbf{ALG}_8$ due to

higher arity [14, pp. 12, 16]. As $\mathbf{ALG}_{11}$ being an almost stream cipher which has relatively low efficiency, as for each two symbols it computes one check symbol, it tends to consume more energy than simpler stream ciphers [14, pp. 18-19].

5. **Scalability** (parallelizability):
   The first five cipher designs $\mathbf{ALG}_1$-$\mathbf{ALG}_5$ are all stream ciphers, this reflects that their design is inherently sequential, which is referred to as that in the algorithm's ciphering process, each symbol depends on the one before it, preventing parallelization within a single message [14, p. 2]. Taking a closer look at $\mathbf{ALG}_1$ and $\mathbf{ALG}_2$, it should be noted that both actually generally scale well for varying message lengths and extending to larger alphabets or longer keys with minimal structural changes [14, pp. 4-5, 19]. $\mathbf{ALG}_3$ and $\mathbf{ALG}_4$ in comparison scale moderately: adding powers of translations/introducing n-ary operations increases computational overhead, so they handle larger messages less efficiently than $\mathbf{ALG}_1$ and $\mathbf{ALG}_2$ [14, pp. 5-7, 19]. Combining both approaches in $\mathbf{ALG}_5$ does not contribute to its general scalability. On the other hand for the block-based designs $\mathbf{ALG}_6$-$\mathbf{ALG}_{11}$, all constructions are parallelizable on block-level, leading us to a general high parallelizability, this especially holds for $\mathbf{ALG}_6$ and $\mathbf{ALG}_7$ [14, pp. 2, 8-10, 19]. Advancing to $\mathbf{ALG}_8$, it has only moderate scalability as $\mathbf{ALG}_8$ is an almost stream cipher [14, p. 12]. Moving along to $\mathbf{ALG}_9$/$\mathbf{ALG}_{10}$, they have high scalability due to their abstract, modular design, allowing easy adaptation to larger data streams and multiple parallel processing units [14, p. 16]. Closing it in now with $\mathbf{ALG}_{11}$, it does come with moderate scalability; its reliance on sequential stream processing and frequent cryptographical term updates limits parallelization compared to fully parallel block ciphers [14, pp. 16-19].

5.3 CBC MAC and its derivations

This paragraph features the results of the comparison of the (One-Key) CBC MACs that were discussed earlier in section 4.2 (see Table 2).

1. **Security** (cryptanalytic resistance):
   The most basic block-based MAC is **CBC MAC**, which has been shown to be secure only for fixed message length mn-bit strings [1] [7, p. 1]. The successor of **CBC MAC** is **EMAC**, which increases security slightly by adding one extra round of encryption using the block-cipher with another key as the final step [13] [7, p. 1]. Finally, **XCBC** establishes security for arbitrary message lengths up to the limit of the birthday paradox by using different keys (3 in total) for padded and respectively non-padded messages [2, pp. 12-15] [7, p. 3]. While **TMAC** and **OMAC** preserve the strong cryptographic properties of **XCBC**, they optimize the algorithms to only require two keys (**TMAC**) [10, pp. 6-9]/one key (**OMAC**) [7, p. 3, 6-12]. Last but not least the most secure MAC in the comparison is **RMAC**. By using a mask of random bits for the key, this algorithm achieves even proven security beyond the birthday paradox limit [8, pp. 8-10] [7, p. 3], therefore it is the strongest MAC in our comparison.
2. **Performance** (throughput/cipher speed):
   Due to its simplicity and minimal design, the **CBC MAC** is very performant [7, p. 1]. As the proposed expansion **EMAC** requires two key schedulings, the performance is worse than default **CBC MAC** [7, p. 1]. The same results hold for **RMAC**: it is a derivation from **EMAC**, so it also requires two keys and schedulings, but additionally it uses random mask and XOR operation. Luckily, PRGs and XOR operations are very fast, so performance it not much slower than original **EMAC** [7, p. 3]. Even that **XCBC** requires three keys in total, the overall performance is better than **EMAC** (only one key schedule required) but still worse than **CBC MAC**, as it requires key lengths of $(k+2n)$-bits in total [7, p. 1]. Finally, **TMAC** and **OMAC** enhance **XCBC** by reducing the number of required keys. This works by deriving the spared keys from a single secret key using efficient multiplications in $\mathrm{GF}(2^n)$, as those operations are computationally very cheap. That said, the throughput and therefore performance of **TMAC** and **OMAC** are pretty balanced, but they are a bit slower than **XCBC** due to slightly increased computational complexity [7, pp. 3, 6].
3. **Complexity** (gate count/silicon area):
   As a circuit implementing **CBC MAC** has no need to realize more than an ordinary block cipher and some cheap XOR operations, this algorithm can be considered to have very low complexity [7, pp. 4, 15]. Moving on with **EMAC**, it can be said that since it performs an extra round of encryption with a distinct key (which doubles key size to $2k$ bits and requires an additional key scheduling, it is noticeably more complex than **CBC MAC** [7, p. 1]. **RMAC** also uses two keys ($2k$ bits), but as it introduces randomness to **EMAC**, a pseudo-random generator is required, which increases gate count even further [7, p. 3]. Coming to **XCBC**, another derivation of **CBC MAC**, possible hardware implementations have to consider the increased key size of $k+2n$ bits in total; besides the handling of block padding, the algorithm is not much more complex than **CBC MAC**, similar to **EMAC**, but with only one key scheduling [7, p. 15]. In comparison, **TMAC** spares one key (reducing key size to $k+n$ bits) by calculating one of the three **XCBC** keys using efficient multiplication in $\mathrm{GF}(2^n)$, which also has to be implemented in hardware [7, p. 2]. Finally, **OMAC** continues reduction of key size down to k bits (like the default **CBC MAC**) by taking full advantage of the al-

Algorithms	Comparison Metrics				
	Security	**Performance**	**Complexity**	**Efficiency**	**Scalability**
CBC MAC	*moderate* resistance; secure for fixed length msgs. of *mn* bits [1] [7, p. 1]	*very high* throughput;	*very low* gate count;	*very low* energy consumption;	*very low* parallelizability;
		most minimal block-based MAC working only for fixed message lengths of (nm bits) [7, p. 1]; short key length (k bits), requires only one key schedule [7, p. 4]; invokes E and $\oplus$ operations $\lvert M\rvert/n$ times in total [7, p. 15]			
EMAC	*moderate* resistance; secure for fixed length msgs. of *mn* bits with extra layer of encryption [13] [7, p. 1]	*moderate* throughput;	*moderate* gate count;	*moderate* energy consumpt.;	*very low* parallelizability;
		EMAC performs another round of E but with another key; increased key length ($2k$ bits), requires two distinct key schedules [7, p. 1]; E is called $1+(\lvert M\rvert/n)$ times in total [7, p. 15]			
RMAC	*very high* resistance; secure for arbitrary message lengths beyond the birthday paradox limit [8, pp. 8-10] [7, p. 3]	*low* throughput;	*high* gate count;	*high* energy consumption;	*very low* parallelizability;
		as **EMAC**, **RMAC** also performs two rounds of E but with a second key but additionally uses randomness [7, p. 3]; doubled key length ($2k$ bits), requires up to $1+\#M$ distinct key schedules, new schedule for each tag [7, p. 15]; E is called $1+\lfloor(\lvert M\rvert+1)/n\rfloor$ times in total [7, p. 15]			
XCBC	*high* resistance; secure for arbitrary message lengths up to birthday paradox limit [2, pp. 12-15] [7, p. 3]	*high* throughput;	*low* gate count;	*low* energy consumption;	*very low* parallelizability;
		even **XCBC** requires three keys in total ($k+2n$ bits), only a single key schedule is needed [7, p. 1-2]; a total of $\lceil\lvert M\rvert/n\rceil$ calls of E are performed [7, p. 15]			
TMAC	*high* resistance; secure for arbitrary message lengths up to birthday paradox limit [10, pp. 6-9] [7, p. 3]	*moderate* throughput;	*moderate* gate count;	*moderate* energy consumption;	*very low* parallelizability;
		TMAC uses efficient multiplication in $\mathrm{GF}(2^n)$ to derive one key, being a tradeoff between reduced key length (to $k+n$ bits) and increased computational complexity [7, p. 2]; otherwise regarding key schedulings (only one) and involved block cipher calls ($\lceil\lvert M\rvert/n\rceil$) its the same as for **XCBC** [7, p. 15]			
OMAC	*high* resistance; secure for arbitrary message lengths up to birthday paradox limit [7, pp. 3, 6-12]	*moderate* throughput;	*moderate* gate count;	*moderate* energy consumption;	*very low* parallelizability;
		OMAC also takes advantage of multiplication in $\mathrm{GF}(2^n)$, but derives two keys to reduce key length to k bits which again leads to increased computational complexity [7, p. 3]; required key schedulings and involved block cipher calls remain the same as for **XCBC** [7, p. 15]			

Table 2: Conceptual comparison of CBC MACs

ready implemented efficient multiplication and deriving two keys; this does not really increase gate count compared to **TMAC**, as implementation in $\mathrm{GF}(2^n)$ is already available [7, p. 6].

4. **Efficiency** (energy consumption):
   Also in terms of energy consumption **CBC MAC** is apparently very efficient, since it only requires a short k-bit key, a single key scheduling and $\lvert M\rvert/n$ calls of the underlying block cipher [7, pp. 4, 15]. As **EMAC** demands two key generations and an additional block cipher round, it also requires more energy to setup and run the MAC [7, p. 1]. For **RMAC**, as it eventually requires multiple key schedulings to underlying block cipher E, it is clear that it needs significantly more energy depending on how many tags to create [7, p. 15]. Regarding **XCBC**, as it just handles padding and delegates its calls to **CBC MAC**, its energy consumption is only slightly above **CBC MAC** [7, pp. 1-2]. Since the implementation of multiplication in $\mathrm{GF}(2^n)$ is straightforwarded, the required energy for **TMAC** only increases a bit compared to **XCBC**. That said, **OMAC** must have very similar energy consumption, as one additional multiplication in $\mathrm{GF}(2^n)$ will have barely any impact (derivation of keys only requires one shift and one conditional $\oplus$ operation) [7, p. 6].
5. **Scalability** (parallelizability):
   The **CBC MAC**, as it already says in the name, uses cipher block chaining (CBC) mode of operation, which by definition does not allow any block-level parallelization [7, p. 1]. The same holds for **EMAC** and **RMAC**, as they just use a **CBC MAC** and encrypt the result with another key, acting strictly sequential only [7, p. 15]. As **XCBC** also uses a chained construction, even **TMAC** and **OMAC** have no possibility for parallelization.

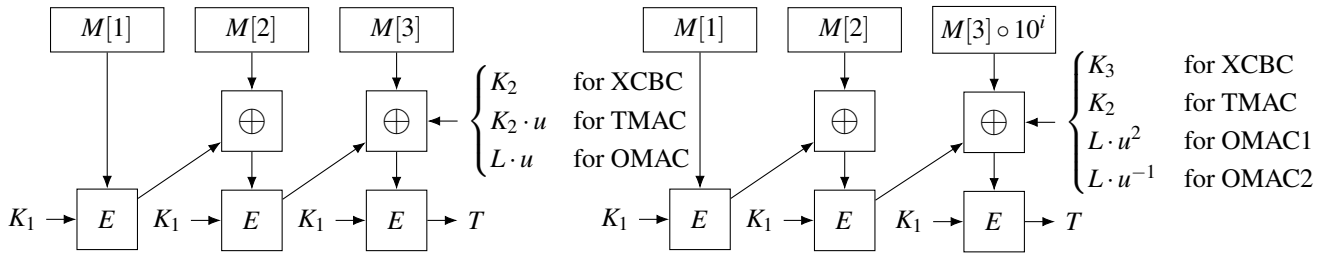

Figure 3: Illustrated comparison of **XCBC** vs. **TMAC** vs. **OMAC** [7, Fig. 1, Fig. 2], where $L = E_{K_1}(0^n)$ and $u \in \mathrm{GF}(2^n)$ is some non-zero constant

5.4 Evaluation

1. **Security** (cryptanalytic resistance):
   quasigroup-based algorithms:
   The algorithms $\mathbf{ALG}_1$, $\mathbf{ALG}_2$ and $\mathbf{ALG}_4$ are not secure and should therefore never be used; the most secure algorithm is $\mathbf{ALG}_{11}$, as it uses an additional cryptcode to detect errors.
   block-based MAC constructions:
   CBC MAC and **EMAC** are only secure for specific message lengths, limiting their usage in real-world applications; while **XCBC**, **TMAC** and **OMAC** are legitimate choices, **RMAC** is especially secure due to additional randomness.
2. **Performance** (throughput/cipher speed):
   quasigroup-based algorithms:
   It is obvious that stream ciphers like $\mathbf{ALG}_1$-$\mathbf{ALG}_5$ have significant higher throughput due to its simplicity than block-based approaches like $\mathbf{ALG}_6$-$\mathbf{ALG}_{11}$.
   block-based MAC constructions:
   CBC MAC is the most simple block-based MAC and therefore it has the highest throughput, but as **XCBC**, **TMAC** and **OMAC** introduce powerful optimizations, elevated throughput can be achieved, only **EMAC** and especially **RMAC** are significantly slower.
3. **Complexity** (gate count/silicon area):
   quasigroup-based algorithms:
   While we know that $\mathbf{ALG}_1$-$\mathbf{ALG}_3$ have very little complexity compared to $\mathbf{ALG}_9$-$\mathbf{ALG}_{11}$, the distinction between $\mathbf{ALG}_4$-$\mathbf{ALG}_8$ is more challenging without actual measurement results (see section 6.2). *block-based MAC constructions*:
   It's obvious that **CBC MAC** has the lowest gate count, also we assume **TMAC** and **OMAC** do not differ that much in their complexity, as the circuits implementing GF(2^n)-multiplication are already available.
4. **Efficiency** (energy consumption):
   quasigroup-based algorithms:
   Regarding energy consumption, we estimated stream ciphers to be very efficient, while block-based constructions have higher energy consumption; increasing arity or block size might lead to less efficient algorithms.
   block-based MAC constructions:
   Nearly all of the discussed MAC designs have little to moderate energy consumption, **RMAC** being an exception as amount of key scheduling scales linearly for each new tag.
5. **Scalability** (parallelizability):
   quasigroup-based algorithms:
   As all stream cipher $\mathbf{ALG}_1$-$\mathbf{ALG}_5$ constructions are inherently sequential by design, they are not parallelizable; in contrast, block-based designs allow at least moderate up to high parallelizability, depending on block size. *block-based MAC constructions*:
   Unfortunately, all constructions are based on **CBC MAC**, which does not allow parallelized block encryption by definition, so explorative research needs to be done (see section 6.2).

6 Summary

This paper explored the use of quasigroup-based cryptographic algorithms in lightweight hardware environments, focusing on their structural simplicity, cryptographic strength, and implementation feasability. Preliminary results indicate that quasigroup-based designs offer promising trade-offs in memory usage and circuit complexity, making them viable candidates for constrained ubiquitous applications, such as RFID systems and low-power IoT nodes. Quasigroup-based primitives have the potential to reduce the circuit's area (gate count/silicon area) by 30-45% (40% reduction compared to lookup-table approach) by heavily reusing circuits, while also decreasing throughput to 25% [12, p. 6-7]. So while we know that quasigroup-based algorithms are much cheaper and consume much less power, they are also much slower, but since throughput is not that important in IoT devices, quasigroup encryption still provides a low-cost solution for embedded systems compared to AES [11, p. 4].

6.1 Results

Starting with quasigroup-based stream cipher algorithms, which share to offer a high throughput and efficiency, relatively low complexity and to not scale well (at least regarding parallelizability) as they are inherently sequential designs:
$\mathbf{ALG}_1$ and $\mathbf{ALG}_2$ being the most simple algorithms in the comparison, have found to be very space-efficient and performant, but since they are proven to be insecure and vulnerable to statistical attacks, they should not be used in any real-world applications. Regarding throughput, the best compromise for high performance while still being security and very efficient is probably $\mathbf{ALG}_3$, as it offers CCA-/CPA-security with only slightly increased computational effort compared to $\mathbf{ALG}_1$/$\mathbf{ALG}_2$. So, whenever resources are strictly limited and real-time processing is necessary, $\mathbf{ALG}_3$ is a good choice. As the generalization of $\mathbf{ALG}_1$/$\mathbf{ALG}_2$ to n-ary operations does not contribute to it's security, $\mathbf{ALG}_4$ also should never be used in real-world applications. $\mathbf{ALG}_5$, as being the most complex stream-based approach and propably offering the best security of the proposed stream ciphers, it seems like a nice fit for embedded systems, where real-time processing is required and at least basic computational capabilities are available.
Continuing with quasigroup-based block encryption schemes, which have generally a higher complexity and therefore require more gates and silicon area. While being slower than quasigroup-based stream ciphers, block-based approaches are also easier to parallelize. $\mathbf{ALG}_6$ is the first cipher which is

based on orthogonal quasigroup structures, which are used to increase the potential permutation set size to $(q^n)!$, so from here, security is guaranteed. Being a sufficiently safe but still very simple block-cipher, $\mathbf{ALG}_6$ might be a potential choice as primitive for **CBC MAC** constructions. In comparison, it is obvious that $\mathbf{ALG}_7$ and $\mathbf{ALG}_8$, as they use combinations of algorithms, have higher complexity, gate count and energy consumption. For $\mathbf{ALG}_7$ the throughput is lower because twice as many calls are required from the underlying $\mathbf{ALG}_6$ compared to $\mathbf{ALG}_8$, which is the reason why $\mathbf{ALG}_8$ (almost stream cipher, therefore slightly increased throughput compared to $\mathbf{ALG}_6$ and $\mathbf{ALG}_7$) was used for further generalization.

The most promising construction in our opinion for potential standardization is $\mathbf{ALG}_9$ with $\mathbf{ALG}_{10}$ as a basic instantiation. Due to the inductive definition of $\mathbf{ALG}_9$, secure algorithms (both stream- and block-based schemes) can be generated that are tailored for specific applications. $\mathbf{ALG}_{10}$ can be seen as n-ary generalization of the binary $\mathbf{ALG}_8$, making it easier for parallelization, so $\mathbf{ALG}_{10}$ is propably the best choice as concrete all-purpose block-cipher. Last but not least, $\mathbf{ALG}_{11}$ turns the cipher into a cryptcode by introducing error-detecting check symbols. Therefore this algorithm is only suitable for very specific applications where, in addition to confidentiality, the integrity of the information must also be absolutely guaranteed, such as in embedded systems for aircraft or critical infrastructure. It should be noted that, beside this algorithm is highly resistant against attacks, it comes with a tradeoff. Throughput, gate count and energy consumption, but also parallelizability are worse, since $\mathbf{ALG}_{11}$ (like $\mathbf{ALG}_8$) is an almost stream-cipher working on pairs of information and the additionally introduced check symbols result in +50% of ciphertext length.

Coming to the discussed MAC algorithms, it is clear that those construction should be build on a quasigroup-based block cipher to further optimize the implementation (we recommend $\mathbf{ALG}_6$ for critical low-resource environments and $\mathbf{ALG}_{10}$ if at least basic computational capabilities are available). Starting with the **CBC MAC**, it is the most minimal block-based MAC, but only works for fixed message lengths, so it is very limited and inappropriate in the most cases, which is why we do not generally recommend to use it. The same basically holds for **EMAC**, as it is only secure for message lengths of multiples of some constant, and because it therefore introduces a whole new extra layer of encryption, key size is doubled and respectively the complexity is also increased. In contrast, **RMAC** is a quite interesting algorithm to consider, because it offers very strong resistance against cryptanalytic attacks and by introducing randomness to the MAC, it is the only algorithm which appears to be secure beyond the birthday paradox limit, a quite unique property. Still, **RMAC** is quite complex as key size is doubled compared to **CBC MAC**, the amount of key schedulings increases linearly by message length and it requires a PRG module, but it is still worth considering if security is definitely the most important requirement. Moving on with XCBC, besides it needs three keys, only one key scheduling is required from this point and most important, it works for message of arbitrary lengths. XCBC should be used if there is no space left to implement an efficient way of multiplication in closed fields, which leads us to **TMAC** and **OMAC**. As **TMAC** and **OMAC** are very similar constructions, there is probably no scenario why one would choose **TMAC** over **OMAC**, since it does not require much more circuitry, as multiplication in closed fields has already been implemented by **TMAC**. So for the discussed MAC algorithms, **OMAC** seems to be the best choice for potential standardization, as it works for arbitrary message lengths with sufficient security properties and a reasonable key size. One problem all of the discussed MAC algorithms share is that there is no way to parallelize the computation on block-level, as they are all based on chained constructions, calculation has to be carried out strictly sequential.

6.2 Future work

First it is clear that there are many more quasigroup-based algorithms available that we could not cover in this work. Therefore we propose to expand the comparison, starting with the following constructions: **Q-S-BOX** (quasigroup-based S-Boxes) [12], **TWINE** and **PRESENT** [11], **INRU** [15] and **SEBQ** [9]; see section 3 for more. As for the discussed MACs, it is obvious that there is no parallelizable scheme in the comparison, therefore it should be extended, e.g. by **PMAC** [3].

During our investigation we realized that comparing the algorithms only based on theoretical material, e.g. papers, is not sufficient to allow a detailed assessment of the algorithms. In most cases there are few to no solid information regarding throughput, gate count, energy consumption or parallelizability, which is the reason why our comparison and the assigned qualitative labels are partly only vague estimates. Therefore we propose to compare actual real implementations of the algorithms by measuring the features quantitatively, which has been beyond the scope of this initial work.

Furthermore, a systematic classification of lightweight cryptographic primitives and algorithms would be a promising direction. This could be based on their underlying design paradigms, such as Substitution-Permutation Networks (SPNs), Feistel structures, or Latin-square-based constructions. Such a classification would help to highlight structural similarities and differences between quasigroup-based and other lightweight schemes. In combination with a rigorous methodology for the experimental evaluation (e.g. benchmarking throughput, gate count, energy consumption and parallelizability), this would enable a more comprehensive and reliable comparison framework for future studies.

References

This work was created independently and was linguistically revised with the help of Grammarly/ChatGPT.

[1] Jee Hea An and Mihir Bellare. Constructing vil-macs from fil-macs: Message authentication under weakened assumptions. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 252–269, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[2] John Black and Phillip Rogaway. Cbc macs for arbitrary-length messages: The three-key constructions. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, page 197–215, Berlin, Heidelberg, 2000. Springer-Verlag.

[3] John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. Cryptology ePrint Archive, Paper 2001/027, 2001.

[4] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 450–466, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[5] Piroska Csorgo and Victor Shcherbacov. On some quasigroup cryptographical primitives, 2011.

[6] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The led block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, pages 326–341, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[7] Tetsu Iwata and Kaoru Kurosawa. Omac: One-key cbc mac. In Thomas Johansson, editor, *Fast Software Encryption*, pages 129–153, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[8] Eliane Jaulmes, Antoine Joux, and Frederic Valette. On the security of randomized CBC-MAC beyond the birthday paradox limit - a new construction. Cryptology ePrint Archive, Paper 2001/074, 2001.

[9] Satish Kumar, Harshdeep Singh, Indivar Gupta, and Ashok Ji Gupta. Symmetric encryption scheme based on quasigroup using chained mode of operation, 2024.

[10] Kaoru Kurosawa and Tetsu Iwata. TMAC: Two-key CBC MAC. Cryptology ePrint Archive, Paper 2002/092, 2002.

[11] William Mahoney, Abhishek Parakh, and Matthew Battey. Hardware implementation of quasigroup encryption for scada networks. In *2014 IEEE 13th International Symposium on Network Computing and Applications*, pages 301–305, 2014.

[12] Hristina Mihajloska, Tolga Yalcin, and Danilo Gligoroski. How lightweight is the hardware implementation of quasigroup s-boxes. In Smile Markovski and Marjan Gusev, editors, *ICT Innovations 2012*, pages 121–128, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[13] Erez Petrank and Charles Rackoff. Cbc mac for real-time data sources. *Journal of Cryptology*, 13(3):315–338, 2000.

[14] Victor Shcherbacov. Quasigroup based crypto-algorithms, 2012.

[15] Sharwan K. Tiwari, Ambrish Awasthi, Sucheta Chakrabarti, and Sudha Yadav. INRU: A quasigroup based lightweight block cipher. *CoRR*, abs/2112.07411, 2021.

[16] Meltem Sönmez Turan, Kerry McKay, Jinkeon Kang, John Kelsey, and Donghoon Chang. Ascon-based lightweight cryptography standards for constrained devices: Authenticated encryption, hash, and extendable output functions. Computer Security Resource Center, 2025.

[17] Milan Vojvoda. *Stream ciphers and hash functions - analysis of some design approachers*. PhD thesis, Fakulta elektrotechniky a informatiky, 2004.