

# Bähēm

## Provably Secure Symmetric Cipher

M. Rajululkahf<sup>1</sup>

May 2, 2022

### Overview

This paper proposes Bähēm; a symmetric cipher that, when given a random-looking key  $\mathbf{k}$ , a true random number generator (TRNG) and a cleartext message  $\mathbf{m}$  to encrypt, no cryptanalysis can degrade its security below  $\min[H(\mathbf{m}), H(\mathbf{k})]$  bits of entropy, even under Grover’s algorithm [1] or even if it turned out that  $P = NP$ .

Aside from the cost of memory access and input/output processing, Bähēm is also highly parallelise-able, and requires only three additions and one bitwise exclusive-or operation (XOR) in order to encrypt or decrypt.

Despite Bähēm’s 2-bits overhead per cleartext bit, its early prototype, Alyal, achieved similar run-time speeds to OpenSSL’s ChaCha20 [2]; slightly faster decryption, while slightly slower encryption when the TRNG was prepared in a file in advance. This demonstrates that Bähēm is practicality usable for many real-world application scenarios.

Later implementations, with better TRNG optimisations and parallelism, must allow the prototype a faster run-time for both, encryption and decryption.

### Notation

$H(\mathbf{x})$ : Shannon’s entropy of random variable  $\mathbf{x}$ .

$\mathbf{x} + \mathbf{y} \bmod 2^{128}$ : Unsigned 128-bit addition.

$\text{random}(128)$ : A sequence of 128 many random bits generated by a TRNG.

$\mathbf{k}$ : A 128-bit pre-shared secret key with enough  $H(\mathbf{k})$  that looks random. Ideally  $\mathbf{k} = \text{random}(128)$ .

$\mathbf{m}$ : A cleartext message of  $|\mathbf{m}|$  many bits.

$\lceil \frac{|\mathbf{m}|}{128} \rceil$ : Number of 128-bit blocks in cleartext  $\mathbf{m}$ .

$\mathbf{m}_b$ : The  $b^{\text{th}}$  128-bit block from  $\mathbf{m}$ . In other words:  $\mathbf{m}_0 \| \mathbf{m}_1 \| \dots \| \mathbf{m}_{\lceil \frac{|\mathbf{m}|}{128} \rceil} = \mathbf{m}$ .

$\mathbf{p}_b = \text{random}(128), \mathbf{q}_b = \text{random}(128)$ : A pair of uniformly distributed 128 many random bits.

$\hat{\mathbf{p}}_b, \hat{\mathbf{q}}_b, \hat{\mathbf{m}}_b$ : Encrypted forms of  $\mathbf{p}_b, \mathbf{q}_b$  and  $\mathbf{m}_b$ , respectively.

### Contents

1	Proposed Algorithm	1
2	Security Analysis	2
3	Implementation Examples	3
3.1	C Functions	3
3.2	A File Encryption Tool	3
3.2.1	Installation	3
3.2.2	Usage	3
3.2.3	Benchmark	3
4	Conclusions	4

## 1 Proposed Algorithm

Algorithms 1 and 2 show Bähēm’s encryption and decryption by which the process is repeated over every 128-bit blocks of  $\mathbf{m}$ :  $\mathbf{m}_0, \mathbf{m}_1, \dots, \mathbf{m}_{\lceil \frac{|\mathbf{m}|}{128} \rceil}$ .

---

#### Algorithm 1: Bähēm encryption

---

**input** :  $\mathbf{k}, \mathbf{m}_0, \mathbf{m}_1, \dots$

**output**:  $(\hat{\mathbf{p}}_0, \hat{\mathbf{q}}_0, \hat{\mathbf{m}}_0), (\hat{\mathbf{p}}_1, \hat{\mathbf{q}}_1, \hat{\mathbf{m}}_1), \dots$

---

```
for  $b \in (0, 1, \dots, \lceil \frac{|\mathbf{m}|}{128} \rceil - 1)$  do
     $\mathbf{p}_b \leftarrow \text{random}(128)$ 
     $\mathbf{q}_b \leftarrow \text{random}(128)$ 
     $\hat{\mathbf{p}}_b \leftarrow \mathbf{p}_b + \mathbf{k} \bmod 2^{128}$ 
     $\hat{\mathbf{q}}_b \leftarrow \mathbf{q}_b + \mathbf{k} \bmod 2^{128}$ 
     $\hat{\mathbf{m}}_b \leftarrow \mathbf{m}_b \oplus (\mathbf{p}_b + \mathbf{q}_b \bmod 2^{128})$ 
```

---

---

#### Algorithm 2: Bähēm decryption

---

**input** :  $\mathbf{k}, (\hat{\mathbf{p}}_0, \hat{\mathbf{q}}_0, \hat{\mathbf{m}}_0), (\hat{\mathbf{p}}_1, \hat{\mathbf{q}}_1, \hat{\mathbf{m}}_1), \dots$

**output**:  $\mathbf{m}_0, \mathbf{m}_1, \dots$

---

```
for  $b \in (0, 1, \dots, \lceil \frac{|\mathbf{m}|}{128} \rceil - 1)$  do
     $\mathbf{p}_b \leftarrow \hat{\mathbf{p}}_b - \mathbf{k} \bmod 2^{128}$ 
     $\mathbf{q}_b \leftarrow \hat{\mathbf{q}}_b - \mathbf{k} \bmod 2^{128}$ 
     $\mathbf{m}_b \leftarrow \hat{\mathbf{m}}_b \oplus (\mathbf{p}_b + \mathbf{q}_b \bmod 2^{128})$ 
```

---

This work is licensed under a Creative Commons “Attribution 4.0 International” license.



<sup>1</sup>Author’s e-mail address: {last name}@pm.me

## 2 Security Analysis

The Bähēm encryption is essentially the XOR cryptosystem:

$$\hat{\mathbf{m}}_b \leftarrow \mathbf{m}_b \oplus \underbrace{(\mathbf{p}_b + \mathbf{q}_b \bmod 2^{128})}_{\text{Encryption pad}}$$

It trivially follows from Shannon’s perfect secrecy proof of the one-time pad (OTP) [3] that Bähēm is secure if and only if its encryption pad maintains its key’s entropy as shown in eq. (1), even if the adversary knows  $\hat{\mathbf{p}}_b$ ,  $\hat{\mathbf{q}}_b$ ,  $\hat{\mathbf{m}}_b$  and the cleartext message  $\mathbf{m}_b$ .

$$\begin{aligned} & H(\mathbf{k} | \hat{\mathbf{p}}_b, \hat{\mathbf{q}}_b, \hat{\mathbf{m}}_b, \mathbf{m}_b) \\ &= H(\mathbf{k} | \hat{\mathbf{p}}_b, \hat{\mathbf{q}}_b, \mathbf{p}_b + \mathbf{q}_b \bmod 2^{128}) \\ &= H(\mathbf{k}) \end{aligned} \quad (1)$$

To simplify the analysis, suppose that the size of a block in Bähēm is 3 bits only, and that the cleartext block  $\mathbf{m}_b$  is known to the adversary, which implies that the adversary can trivially know that:

$$\mathbf{p}_b + \mathbf{q}_b \bmod 2^3 = \hat{\mathbf{m}}_b \oplus \mathbf{m}_b$$

in addition to adversary’s knowledge of the public variables  $\hat{\mathbf{p}}_b$  and  $\hat{\mathbf{q}}_b$ . More specifically, suppose that the adversary found that:

$$\begin{aligned} 0 &= \hat{\mathbf{p}}_b = \mathbf{p}_b + \mathbf{k} \bmod 2^3 \\ 3 &= \hat{\mathbf{q}}_b = \mathbf{q}_b + \mathbf{k} \bmod 2^3 \\ 5 &= \mathbf{p}_b + \mathbf{q}_b \bmod 2^3 \end{aligned}$$

Then, the question is: will this information reduce the space from which the key  $\mathbf{k}$  is chosen from? In other words, what are the possible values of  $\mathbf{k}$  that can lead to the outputs 0, 3 and 5 above? Table 1 visualises this.

As shown in table 1, the total number of horizontal, or vertical, intersections that simultaneously cross all of the outputs 0, 3 and 5, remain  $2^3$ . Meaning, the total number of values of  $\mathbf{k}$  that could lead to the outputs remains  $2^3$ .

This 3-bit example can be trivially extended by induction to show that the same conclusions hold even with a 128-bit unsigned addition and any other output numbers than 0, 3 and 5.

Therefore, we can conclude that adversary’s knowledge of the public variables  $\hat{\mathbf{p}}_b$ ,  $\hat{\mathbf{q}}_b$ ,  $\hat{\mathbf{m}}_b$  and the cleartext  $\mathbf{m}_b$ , which leads to deducing  $\mathbf{p}_b + \mathbf{q}_b \bmod 2^{128}$ , can not reduce the space from which  $\mathbf{k}$ ,  $\mathbf{p}_q$  and  $\mathbf{q}_q$  are sampled.

	$\mathcal{Y}$							
	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Table 1: Exhaustive unsigned 3-bit addition. For a given output  $\mathbf{x} + \mathbf{y} \bmod 2^3$ , there are  $2^3$  many possible input values of  $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$  that map to  $\mathbf{x} + \mathbf{y} \bmod 2^3$ .

If  $\mathbf{k}$ ,  $\mathbf{p}_q$  and  $\mathbf{q}_q$  are generated by a TRNG, then any of the  $2^{128}$  many possibilities are equally likely to correspond to the actual values of  $\mathbf{k}$ ,  $\mathbf{p}_q$  and  $\mathbf{q}_q$ . In other words:

$$H(\mathbf{k}, \mathbf{p}_q, \mathbf{q}_q | \hat{\mathbf{p}}_b, \hat{\mathbf{q}}_b, \hat{\mathbf{m}}_b, \mathbf{m}_b) = 128$$

However, since  $\mathbf{k}$  could be derived from a password, such that it looks random, but with an entropy  $H(\mathbf{k}) \leq 128$ , and since finding any of the numbers  $\mathbf{k}$ ,  $\mathbf{p}_q$  and  $\mathbf{q}_q$  deterministically leads to finding the others, therefore it follows that:

$$H(\mathbf{k}, \mathbf{p}_q, \mathbf{q}_q | \hat{\mathbf{p}}_b, \hat{\mathbf{q}}_b, \hat{\mathbf{m}}_b, \mathbf{m}_b) = H(\mathbf{k})$$

The numbers  $\mathbf{p}_q$  and  $\mathbf{q}_q$  are generated by a TRNG by definition, therefore the weakest element in the chain can be only  $\mathbf{k}$ .

Since the public variables  $\hat{\mathbf{p}}_b$ ,  $\hat{\mathbf{q}}_b$ ,  $\hat{\mathbf{m}}_b$  and the cleartext  $\mathbf{m}_b$  are exhaustively all of the outputs of Bähēm that can be accessible to an adversary, and since they can not reduce Bähēm’s parameters space below  $H(\mathbf{k})$ , therefore no cryptanalysis can reduce the entropy Bähēm’s private variables below  $H(\mathbf{k})$ .

**Lemma 1** (Secure private values).

$$H(\mathbf{k}, \mathbf{p}_q, \mathbf{q}_q | \hat{\mathbf{p}}_b, \hat{\mathbf{q}}_b, \hat{\mathbf{m}}_b, \mathbf{m}_b) = H(\mathbf{k})$$

It is trivially implied from lemma 1 that, since the private values  $\mathbf{p}_b$  and  $\mathbf{q}_b$  maintain an entropy of  $H(\mathbf{k})$ , so does their 128-bit summation  $\mathbf{p} + \mathbf{q} \bmod 2^{128}$ , which is Bähēm’s XOR encryption pad. Therefore, Bähēm’s encryption pad has to be secure as well.

**Lemma 2** (Secure encryption pad).

$$H(\mathbf{p} + \mathbf{q} \bmod 2^{128} | \hat{\mathbf{p}}_b, \hat{\mathbf{q}}_b, \hat{\mathbf{m}}_b) = H(\mathbf{k})$$

Since Bāhēm is an XOR cryptosystem, and since its encryption pad is  $H(\mathbf{k})$ -bits secure (lemma 2), therefore it necessarily follows by Shannon’s perfect secrecy [3] that Bāhēm’s encryption is either  $H(\mathbf{k})$ -bits secure, or  $H(\mathbf{m}_b)$ -bits secure, whichever is smaller.

**Theorem 1** (Secure encryption).

$$H(\mathbf{m}_b | \hat{\mathbf{p}}_b, \hat{\mathbf{q}}_b, \hat{\mathbf{m}}_b) = \min[H(\mathbf{m}_b), H(\mathbf{k})]$$

**Note:** Bāhēm does not aim at achieving perfect secrecy, as perfect secrecy requires a usually-impractical key that is as long as the length  $|\mathbf{m}|$  of the cleartext message  $\mathbf{m}$  itself, with an unnecessarily too large upper security bound that is worth  $|\mathbf{m}|$  many bits of entropy.

Bāhēm rather aims at achieving a security that is worth  $\min[H(\mathbf{m}_b), H(\mathbf{k})] \leq 128$  entropy bits. Unlike the OTP, this is *practically* secure and only requires a small 128-bit pre-shared key.

Shannon’s proof of perfect secrecy of the OTP is cited only for its relevance as an XOR cryptosystem.

## 3 Implementation Examples

### 3.1 C Functions

In listing 1, the caller is expected to initialise a 128 bits key  $\mathbf{k}$ , a pair of random pads,  $\mathbf{p}$  and  $\mathbf{q}$ , each of which is `len` long, in order to encrypt a `len` long cleartext message  $\mathbf{m}$ . The encryption happens in-place, so the caller does not have to allocate separate memory for the output. Since 128-bit wide CPU instructions are not common, this code operates in 64-bit basis, each time with a different 64-bit part of the key.

Likewise, listing 2 shows an example implementing the corresponding in-place decryption function.

### 3.2 A File Encryption Tool

Alyal is an single-threaded implementation to demonstrate Bāhēm’s practical utility with real-world scenarios. Internally, Alyal uses the `baheem_enc` and `baheem_dec` functions that were presented earlier in this section.

#### 3.2.1 Installation

```
git clone \
https://codeberg.org/rajululkahf/alyal
cd alyal
make
make test
```

Listing 1: Encryption function example

```
void baheem_enc(
    uint64_t *k, /* 128bit pre-shared key */
    uint64_t *p, /* random pad 1 */
    uint64_t *q, /* random pad 2 */
    uint64_t *m, /* message */
    size_t len /* length of m = p = q */
) {
    size_t i;
    for (i = 0; i < len; i += 2) {
        m[i] ^= p[i] + q[i];
        m[i+1] ^= p[i+1] + q[i+1];
        p[i] += k[0];
        q[i] += k[0];
        p[i+1] += k[1];
        q[i+1] += k[1];
    }
}
```

Listing 2: Decryption function example

```
void baheem_dec(... same input ...) {
    size_t i;
    for (i = 0; i < len; i += 2) {
        p[i] -= k[0];
        q[i] -= k[0];
        p[i+1] -= k[1];
        q[i+1] -= k[1];
        m[i] ^= p[i] + q[i];
        m[i+1] ^= p[i+1] + q[i+1];
    }
}
```

#### 3.2.2 Usage

```
alyal (enc|dec) IN OUT [TRNG]
alyal help
```

To encrypt a cleartext file `a` and save it as file `b`:

```
alyal enc a b
```

To decrypt the latter back to its cleartext form and save it as file `c`:

```
alyal dec b c
```

#### 3.2.3 Benchmark

This is a benchmark that was performed on a computer with a 3.4GHz Intel Core i5-3570K CPU, 32GB RAM, 7200 RPM hard disks, Linux 5.17.4-gentoo-x86-64, and OpenSSL 1.1.1n.

	OpenSSL ChaCha20	Alyal Bähēm	
		/dev/random	file.rand
Encrypt	<b>0.87</b> secs	3.91 secs	1.40 secs
500MB	<b>1.04</b> secs	4.25 secs	1.82 secs
	<b>1.04</b> secs	4.27 secs	1.73 secs
Decrypt	0.90 secs	<b>0.64</b> secs	
500MB	1.06 secs	<b>0.89</b> secs	
	1.06 secs	<b>0.81</b> secs	

Table 2: Wall-clock run-time comparison between OpenSSL’s ChaCha20, and Alyal’s Bähēm implementation with two sources as the TRNG: `/dev/random` and `file.rand`; the latter is simply `/dev/random` that was prepared in advance.

Table 2 shows that, while the early Bähēm prototype, Alyal, has a faster decryption run-time than OpenSSL’s ChaCha20, it has slower encryption run-time. However:

1. The differences in run-time are insignificant for most applications, which proves Bähēm’s practical utility in the real world.
2. Bähēm’s provable security should arguably justify waiting the 3 extra seconds for the 500MB data, specially that many user applications involve encrypting much smaller data sizes with unnoticeable time difference
3. Preparing the random bits in advance significantly reduces the encryption time as shown with the `file.rand` case in table 2, and can be optimised further should it be prepared in memory.
4. Alyal is currently single-threaded despite Bähēm’s capacity for high parallelism as all blocks are independent. This gives room for future versions to be significantly faster.

## 4 Conclusions

This paper proposed Bähēm with the following properties:

**Secure:** No cryptanalysis can degrade its security below  $\min[H(\mathbf{m}), H(\mathbf{k})]$  bits.

**Fast:** Requires only three additions and a single XOR per encryption or decryption alike.

Highly parallelisable as the encryption, or decryption, of any bit is independent of other bits.

Bähēm’s single-threaded prototype, Alyal, outperformed OpenSSL’s ChaCha20 when decrypting files, despite Bähēm’s 2 bits overhead, which

demonstrates that such overhead is negligible in practice.

While the prototype has a slower encryption run-time due to its use of a TRNG, optimising it is trivial by preparing the TRNG in advance.

**Simple:** Bähēm’s simplicity implies fewer expected number of implementation bugs, and therefore higher practical security.

Another interesting advantage of this simplicity is that it allows Bähēm to be used with a mere pen and a paper should one lack a computer, such as the case with post-apocalyptic scenarios.

For example, in a post-apocalyptic scenario, one can generate the random numbers  $\mathbf{p}_b$  and  $\mathbf{q}_b$  by rolling dies enough number of times until adequate entropy is obtained, and then using a pen and a paper to calculate the ciphertext as per algorithm 1.

Since Bähēm does not require repeating rounds over and over, Bähēm is significantly simpler to perform using a pen and a paper than, say, ChaCha20, AES [4], etc, which require many repeated rounds that make it too tedious for a human to perform by the pen and paper method.

## Declarations

All data generated or analysed during this study are included in this published article. Implementation code is available in the following repository:

<https://codeberg.org/rajululkahf/alyal>

## References

- [1] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC ’96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.
- [2] Daniel Bernstein. ChaCha, a variant of Salsa20. 01 2008.
- [3] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, 1949.
- [4] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael, 1999.