


Ġasaq

Provably Secure Key Derivation

M. Rajululkahf ¹

April 21, 2022

Update—Ġasaq is not secure at all. I did many silly mistakes. This paper rightfully got an immediate rejection rejection from the Journal of Cryptology.

Overview

This paper proposes Ġasaq; a provably secure key derivation method that, when given access to a true random number generator (TRNG), allows communicating parties, that have a pre-shared secret password \mathbf{p} , to agree on a secret key \mathbf{k} that is indistinguishable from truly random numbers with a guaranteed entropy of $\min(\mathbf{H}(\mathbf{p}), |\mathbf{k}|)$.

Ġasaq’s security guarantees hold even in a post-quantum world under Grover’s algorithm [1], or even if it turns out that $\mathbf{P} = \mathbf{NP}$ [2]. Such strong security guarantees, that are similar to those of the one-time pad (OTP), became attractive after the introduction of Bāhēm [3]; a similarly provably secure symmetric cipher that is strong enough to shift cipher’s security bottleneck to the key derivation function.

State of art key derivation functions such as the Password-Based Key Derivation Function (PBKDF) [4], or even memory-hard variants such as Argon2 [5], are not provably secure, but rather not fully broken yet. They do not guarantee against needlessly losing password entropies; that is, the output key could have an entropy lower than password’s entropy, even if such entropy is less than key’s bit length. In addition to assuming that $\mathbf{P} \neq \mathbf{NP}$, and, even then, getting their key space square-rooted under Grover’s algorithm —none of which are limitations of Ġasaq.

Using such key derivation functions, as the PBKDF or Argon2, is acceptable with conventional ciphers, such as ChaCha20 [6] or AES [7], as they, too, suffer the same limitations, hence none of them are bottlenecks for the other. Similarly to how a glass door is not a security bottleneck for a glass house.

However, a question is: why would a people secure their belongings in a glass made structure, to justify a glass door, when they can use a re-enforced steel structure at a similar cost? This is where Ġasaq comes to offer Bāhēm the re-enforced steel door that matches its security.

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

Notation

$\text{random}(n) = (r_0, r_1, \dots, r_n)$: A sequence of n many random bits generated by a TRNG.

$\mathbf{x} = (x_0, x_1, \dots, x_{|\mathbf{x}|-1})$: A tuple of $|\mathbf{x}|$ many bits.

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

$\mathbf{x} \oplus \mathbf{y}$: Bitwise exclusive-or operation between two variables.

$\mathbf{p} = (p_0, p_1, \dots, p_{|\mathbf{p}|-1})$: An arbitrarily long pre-shared secret password of $|\mathbf{p}|$ many bits. Since passwords are typed by humans, they often contain redundancies, hence $\mathbf{H}(\mathbf{p}) \leq |\mathbf{p}|$.

$\mathbf{k} = (k_0, k_1, \dots, k_{|\mathbf{k}|-1})$: A secret key derived based on \mathbf{p} .

$\mathbf{q}_i = (p_i, p_i, \dots)$: A tuple containing the i^{th} bit of password \mathbf{p} repeated $|\mathbf{k}|$ many times. In other words, $|\mathbf{q}_i| = |\mathbf{k}|$ and $\mathbf{H}(\mathbf{q}_i) = \mathbf{H}(p_i) \leq 1$.

$\mathbf{r}_i = \text{random}(|\mathbf{k}|)$: A uniformly distributed $|\mathbf{k}|$ bits random number. There are $\mathbf{r}_0, \mathbf{r}_1, \dots, \mathbf{r}_{|\mathbf{p}|-1}$ of such random numbers.

$\hat{\mathbf{r}}_i$: Encrypted form of \mathbf{r}_i that is shared publically.

$\min(n, m)$:

$$\begin{cases} n & \text{if } n < m \\ m & \text{otherwise} \end{cases}$$

Contents

1	Background	2
2	Proposed Algorithm: Ġasaq	2
3	Security Proof	3
3.1	Safe Public Output	3
3.2	Maximum Key Entropy	3
3.3	Random-looking Key	3
4	Implementation Example	4
4.1	An Early Prototype: Alyal	4
4.1.1	Installation and Usage	4
4.1.2	Benchmark	4
5	Conclusion	4
A	XOR Cryptosystem Review	5
B	Examples in C	6

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



1 Background

In a previous private meeting between Alice and Bob, they concluded to use a TRNG to generate 128 random bits, and to use them as their pre-shared secret key. They also agreed to use the Bāhēm symmetric cipher [3]. Further details about this previous meeting can be found in the background section in [3].

However, since then, Alice and Bob found that they are unable to memorise those 128 true random bits, not even with mnemonic encodings. Practically, this made them write the 128 bits in some form of storage, and carried the storage with them.

While carrying the 128 bits in a small storage device is much more manageable than carrying terabytes worth of random one-time pad, it was still not ideal for Alice and Bob.

Their main reasoning is that, if they were to be captured by an adversary, and found to poses the 128 random bits, then their attempts to deny their ownership of encrypted data might become harder. Specially if the adversary found that the 128 bits decrypt some files that they possess. They also had secondary reasons, such as the inconvenience that is associated with backup recovery plans should the device be lost.

However, because of the past memories of Alice and Bob, they could easily memorise odd sentences like:

“The anteater ate steel with his cosmic buddy; the turbo-charged flying octopus”

While Alice and Bob could not calculate the entropy of that sentence, as they lack knowledge about the probabilistic language model inside their own heads, they were nonetheless highly confident that such a phrase has “high enough” entropy for their purpose, specially if it allows them to get rid of the constraint of carrying a password storage device with them.

However, they found that today’s state-of-art key derivation methods, such as PBKDF [4], or even memory-hard variants such as Argon2 [5], are not provably secure; that is, there might be some cryptanalysis techniques that could possibly reduce the entropy of their derived keys. This bothered them, as using a non-proven key derivation function might serve as the Achilles’ heel in their provably secure Bāhēm cipher.

Ġasaq, the key derivation method that is proposed in this paper, solves the Alice and Bob problem above, by offering a provably secure key derivation method similar to that of the OTP, or Bāhēm, that is independent to whether quantum computers were used, or to whether it turned out that $P = NP$.

2 Proposed Algorithm: Ġasaq

Algorithm 1: Ġasaq key creation

```

input :  $\mathbf{p}, |\mathbf{k}|$ 
output:  $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}, \mathbf{k}$ 


---


 $\mathbf{k} \leftarrow (0, 0, \dots)$  a tuple of  $|\mathbf{k}|$  many zeros
for  $i \in \{0, 1, \dots, |\mathbf{p}| - 1\}$  do
   $\mathbf{r}_i \leftarrow \text{random}(|\mathbf{k}|)$ 
   $\mathbf{q}_i \leftarrow (p_i, p_i, \dots)$ , where  $|\mathbf{q}_i| = |\mathbf{k}|$ 
   $\mathbf{k} \leftarrow \mathbf{k} \oplus \mathbf{r}_i$ 
   $\hat{\mathbf{r}}_i \leftarrow \mathbf{r}_i \oplus \mathbf{q}_i$ 
return  $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}, \mathbf{k}$ 

```

Algorithm 2: Ġasaq key retrieval

```

input :  $\mathbf{p}, \hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}$ 
output:  $\mathbf{k}$ 


---


 $\mathbf{k} \leftarrow (0, 0, \dots)$  a tuple of  $|\mathbf{k}|$  many zeros
for  $i \in \{0, 1, \dots, |\mathbf{p}| - 1\}$  do
   $\mathbf{q}_i \leftarrow (p_i, p_i, \dots)$ , where  $|\mathbf{q}_i| = |\mathbf{k}|$ 
   $\mathbf{r}_i \leftarrow \hat{\mathbf{r}}_i \oplus \mathbf{q}_i$ 
   $\mathbf{k} \leftarrow \mathbf{k} \oplus \mathbf{r}_i$ 
return  $\mathbf{k}$ 

```

The key derivation in Ġasaq is in two parts:

Creation. At first, a key \mathbf{k} is defined from the input password \mathbf{p} and a one-time pad of secret random numbers $\mathbf{r}_0, \mathbf{r}_1, \dots, \mathbf{r}_{|\mathbf{k}|-1}$. A corresponding public sequence $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{k}|-1}$ is also created and shared publicly to allow for retrieving the key \mathbf{k} later by the communicating parties that know the pre-shared secret password \mathbf{p} . The key \mathbf{k} itself is not transmitted, but rather found indirectly during the retrieval process.

Key creation is shown in Algorithm 1. Appendix B shows an example implementation in the C programming language.

Retrieval. Using a public sequence $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{k}|-1}$ and its corresponding pre-shared secret password \mathbf{p} , its corresponding private sequence $\mathbf{r}_0, \mathbf{r}_1, \dots, \mathbf{r}_{|\mathbf{k}|-1}$ is retrieved, which is then used to define the secret key \mathbf{k} .

Key retrieval is shown in Algorithm 2. Appendix B shows an example implementation in the C programming language.

In other words, if Alice wants to send Bob an encrypted message by a key that is derived securely

from their pre-shared secret password, she will first send him $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}$ in order to let him securely retrieve of the key, for as long as he knows the pre-shared secret password \mathbf{p} . This key can then be used with a provably secure symmetric cipher, such as Bähēm [3], to encrypt their future messages.

The sequence $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}$ is shareable publicly. That is, Eve seeing it will not reveal to her any information about the password. Yet, the sequence allows Alice and Bob to mutually agree on a perfectly secure key \mathbf{k} based on their pre-shared password \mathbf{p} .

Alice and Bob do not have to exchange the public sequence $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}$ every time they communicate, as they can simply cache the key \mathbf{k} and re-use it for future communications. They will have to repeat the creation-retrieval process again only if they lose the key, or change their password.

3 Security Proof

Definition 3.1. *A key derivation system is perfectly secure if it simultaneously satisfies the following:*

Safe public output: *Knowledge about the public output $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}$ must not reveal information about the password \mathbf{p} or the derived key \mathbf{k} .*

$$\begin{aligned} H(\mathbf{p}|\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}) &= H(\mathbf{p}) \\ H(\mathbf{k}|\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}) &= \min(H(\mathbf{p}), |\mathbf{k}|) \end{aligned}$$

Maximum key entropy: *Entropy of the derived key \mathbf{k} must equate the entropy of the password \mathbf{p} , or the number of key bits $|\mathbf{k}|$, whichever is smaller.*

$$H(\mathbf{k}) = \min(H(\mathbf{p}), |\mathbf{k}|)$$

Random-looking key: *The distribution of bits in \mathbf{k} must be indistinguishable from that of a $|\mathbf{k}|$ many bits of a true random number $\text{random}(|\mathbf{k}|)$.*

3.1 Safe Public Output

Proof. By the definition of Ġasaq, for any password bit $i \in \{0, 1, \dots, |\mathbf{p}| - 1\}$, $\hat{\mathbf{r}}_i$ can be viewed as the ciphertext of the bitwise exclusive-or operation (XOR) cryptosystem $\hat{\mathbf{r}}_i \leftarrow \mathbf{r}_i \oplus \mathbf{q}_i$ where a true random number \mathbf{r}_i is encrypted by \mathbf{q}_i , which is an $|\mathbf{k}|$ many bits repetition of the i^{th} bit of the password \mathbf{p} .

Based on the properties of the XOR cryptosystem, which is described in Appendix A, it follows that, an adversary that observes the public outputs

$\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}$, will not be able to gain any information about the password \mathbf{p} . In other words, $H(\mathbf{p}|\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}) = H(\mathbf{p})$ is guaranteed by the properties of the XOR cryptosystem.

Additionally, since the key \mathbf{k} is defined by XORing the cleartext random numbers $\mathbf{r}_0, \mathbf{r}_2, \dots, \mathbf{r}_{|\mathbf{p}|-1}$, and since information about the password \mathbf{p} is required in order to retrieve those cleartext random numbers from the public output $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_2, \dots, \hat{\mathbf{r}}_{|\mathbf{p}|-1}$, it follows that knowing the public output does not reduce key's entropy below $\min(H(\mathbf{p}), |\mathbf{k}|)$. ■

Theorem 3.1. *Ġasaq has safe public output.*

3.2 Maximum Key Entropy

Proof. Since:

- \mathbf{r}_i is a true $|\mathbf{k}|$ many bits random number by definition, each of which is guaranteed to satisfy $H(\mathbf{r}_i) = H(p_i)$ even when the public output $\hat{\mathbf{r}}_i$ is revealed, as explained in Appendix A.
- \mathbf{k} is the result of XORing $|\mathbf{p}|$ many such random numbers.
- $|\mathbf{k}|$ is usually large enough, such as 128 bits, to have near-zero probability of collisions between the \mathbf{r}_i true numbers.

For example, in order to have birthday collision probability of 10^{-18} with 128 bit random numbers, one would have to generate 2.6×10^{10} many such 128 random bits [8]. Practically, it is trivial to set $|\mathbf{k}|$ to be large enough so that such birthday collisions never happen in practice, while still retaining a usable cryptographic system.

Then it follows that:

$$H(\mathbf{k}) = \begin{cases} H(p_0) + H(p_1) + \dots & \text{if } H(\mathbf{p}) < |\mathbf{k}| - \epsilon \\ |\mathbf{k}| - \epsilon & \text{otherwise} \end{cases}$$

where ϵ is information loss due to the collisions that may arise between $|\mathbf{p}|$ many $|\mathbf{k}|$ long random bits. Since $|\mathbf{k}|$ is usually large enough, $\epsilon \approx 0$. ■

Theorem 3.2. *Ġasaq has maximum key entropy.*

3.3 Random-looking Key

Proof. Since the key \mathbf{k} is defined by XORing true random numbers $\mathbf{r}_0, \mathbf{r}_1, \dots, \mathbf{r}_{|\mathbf{p}|-1}$, it follows by the properties of the XOR cryptosystem that \mathbf{k} is indistinguishable from a true random number. ■

Theorem 3.3. *Ġasaq's keys are indistinguishable from true random numbers.*

4 Implementation Example

4.1 An Early Prototype: Alyal

Alyal is an early single-threaded prototype implementation for securely encrypting files. It uses Ġasaq to derive keys, which are then used to encrypt and decrypt files by the Bāhēm cipher.

Alyal assumes that $|\mathbf{k}| = 128$ bits, and attaches the public random bits $\hat{\mathbf{r}}_0, \hat{\mathbf{r}}_1, \dots, \hat{\mathbf{r}}_{|\mathbf{P}|-1}$ along the ciphertext of every file that it encrypts. Because of Ġasaq’s and Bāhēm’s provable security that is very similar to that of the OTP, $|\mathbf{k}| = 128$ is more than enough, even for a post-quantum world, or even if $\mathbf{P} = \mathbf{NP}$.

4.1.1 Installation and Usage

```
> git clone \
  https://codeberg.org/rajululkahf/alyal
> cd alyal
> make
> dd bs=1MB count=500 \
  if=/dev/zero of=test.txt
> ./alyal dkenc test.txt test.enc
> ./alyal dkdec test.enc test.enc.txt
> shasum *
```

4.1.2 Benchmark

Table 1 shows an early benchmark that was performed on a machine with a 3.4GHz Intel Core i5-3570K CPU, 32GB RAM, 7200 RPM hard disks and Linux 5.17.1-gentoo-x86-64.

	Alyal Raw key	Alyal Ġasaq key	Difference
Encrypt	3.84 secs	3.95 secs	-0.11 secs
500MB	4.29 secs	4.40 secs	-0.11 secs
	4.23 secs	4.34 secs	-0.10 secs
Decrypt	0.66 secs	0.67 secs	-0.01 secs
500MB	0.82 secs	0.82 secs	0.00 secs
	0.82 secs	0.83 secs	-0.01 secs

Table 1: Wall-clock run-time comparison between Alyal’s repeated file encryptions. *Raw key* is when Alyal was given a 128 bit key directly. *Ġasaq* is when Ġasaq was used to derive a key from a password.

Table 1 shows that Ġasaq’s key creation and retrieval overhead is negligible for most applications, even without caching.

5 Conclusion

This paper presented Ġasaq, a provably secure key derivation that guarantees that no cryptanalysis can degrade its security below $\min(H(\mathbf{p}), |\mathbf{k}|)$ many entropy bits with properties identical to that of the OTP, and a negligible overhead for most applications. If Ġasaq’s output is cached, then it has no overhead.

An early implementation of Ġasaq is also released alongside this paper to demonstrate its applicability for real world scenarios, even when Ġasaq’s output is not cached.

Future work can research provably-secure memory-hard variations of Ġasaq-based key derivation methods.

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] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC ’71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.
- [3] M. Rajululkahf. Bahem: A Provably Secure Symmetric Cipher. In *Engineering Archive*, 2022. <https://engrxiv.org/preprint/view/2278>
<https://codeberg.org/rajululkahf/baheem>.
- [4] Kathleen Moriarty, Burt Kaliski, and Andreas Rusch. PKCS #5: Password-Based Cryptography Specification Version 2.1. RFC 8018, January 2017.
- [5] Jos Wetzels. Open sesame: The password hashing competition and argon2. *CoRR*, abs/1602.03097, 2016.
- [6] Daniel Bernstein. Chacha, a variant of salsa20. 01 2008.
- [7] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael, 1999.
- [8] Wikipedia contributors. Birthday attack — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/?oldid=1068584536>, 2022. [Online; accessed 17-April-2022].

A XOR Cryptosystem Review

Since Ġasaq makes frequent use of the XOR cryptosystem, it makes sense to introduce some of its properties first.

Let $(x, r) \in \{0, 1\}^2$ be a pair of independent random variables, with $\Pr(r = 0) = 2^{-1}$, used in the cryptosystem $\hat{x} \leftarrow x \oplus r$, where \hat{x} is the ciphertext.

If the adversary finds the public information that $\hat{x} = 0$, then she can conclude that $x = r$ and deduce the following about the probability $\Pr(x = 0|x = r)$:

$$\begin{aligned}
 &= \frac{\Pr(x = 0) \Pr(x = r|x = 0)}{\Pr(x = r)} \\
 &= \frac{\Pr(x = 0) \Pr(r = 0)}{\Pr(x = 0) \Pr(r = 0) + \Pr(x = 1) \Pr(r = 1)} \\
 &= \frac{\Pr(x = 0) 2^{-1}}{\Pr(x = 0) 2^{-1} + \Pr(x = 1) 2^{-1}} \\
 &= \frac{\Pr(x = 0)}{\Pr(x = 0) + \Pr(x = 1)} \\
 &= \frac{\Pr(x = 0)}{\Pr(x = 0) + (1 - \Pr(x = 0))} \\
 &= \Pr(x = 0)
 \end{aligned}$$

Likewise, the following can be deduced about the probability $\Pr(r = 0|x = r)$:

$$\begin{aligned}
 &= \frac{\Pr(r = 0) \Pr(x = r|r = 0)}{\Pr(x = r)} \\
 &= \frac{\Pr(r = 0) \Pr(x = 0)}{\Pr(x = 0) \Pr(r = 0) + \Pr(x = 1) \Pr(r = 1)} \\
 &= \frac{2^{-1} \Pr(x = 0)}{\Pr(x = 0) 2^{-1} + \Pr(x = 1) 2^{-1}} \\
 &= \frac{\Pr(x = 0)}{\Pr(x = 0) + \Pr(x = 1)} \\
 &= \frac{\Pr(x = 0)}{\Pr(x = 0) + (1 - \Pr(x = 0))} \\
 &= \Pr(x = 0)
 \end{aligned}$$

The same also applies if $\hat{x} = 1$, by which $\Pr(x = 0|x \neq r) = \Pr(r = 0|x \neq r) = \Pr(x = 0)$.

If r was the encryption key, and x was the cleartext, then $\Pr(x = 0|x = r) = \Pr(x = 0)$ implies that the cryptosystem system has perfect secrecy, as the adversary gains no information about the cleartext by knowing the ciphertext \hat{x} . This can be expressed

by using the information gain measure:

$$\begin{aligned}
 IG(x|\hat{x} = 0) &= H(x) - H(x|\hat{x} = 0) \\
 &= \left(\begin{array}{l} \Pr(x = 0) \log_2(\Pr(x = 0)^{-1}) \\ +(1 - \Pr(x = 0)) \log_2((1 - \Pr(x = 0))^{-1}) \\ - \Pr(x = 0) \log_2(\Pr(x = 0)^{-1}) \\ -(1 - \Pr(x = 0)) \log_2((1 - \Pr(x = 0))^{-1}) \end{array} \right) \\
 &= 0
 \end{aligned}$$

On the other hand, if x was the encryption key, and r was the cleartext, then $\Pr(r = 0|x = r) = \Pr(x = 0)$ implies that the cryptosystem system does not have perfect secrecy, as the adversary is able to gain information about the cleartext by knowing the ciphertext \hat{x} . The following quantifies the number of information bits that the adversary gains about the cleartext in this scenario from the ciphertext:

$$\begin{aligned}
 IG(r|\hat{x} = 0) &= H(r) - H(r|\hat{x} = 0) \\
 &= \left(\begin{array}{l} 2^{-1} \log_2(2) \\ +(1 - 2^{-1}) \log_2((1 - 2^{-1})^{-1}) \\ - \Pr(x = 0) \log_2(\Pr(x = 0)^{-1}) \\ -(1 - \Pr(x = 0)) \log_2((1 - \Pr(x = 0))^{-1}) \end{array} \right) \\
 &= \left(\begin{array}{l} 1 \\ - \Pr(x = 0) \log_2(\Pr(x = 0)^{-1}) \\ -(1 - \Pr(x = 0)) \log_2((1 - \Pr(x = 0))^{-1}) \end{array} \right) \\
 &= 1 - H(x)
 \end{aligned}$$

In other words, $IG(r|\hat{x} = 0) = 1 - H(x)$ means that the information about x is preserved, and everything else is lost to the adversary. This also implies that $H(r) = H(x)$.

For any $i \in \{0, 1, \dots, |\mathbf{p}| - 1\}$, the reason behind Ġasaq's use of the cryptosystem $\hat{\mathbf{r}}_i \leftarrow \mathbf{r}_i \oplus \mathbf{q}_i$ is not to preserve all information about \mathbf{r}_i , but rather to ensure that $H(\mathbf{r}_i) = H(\mathbf{q}_i) = H(p_i)$, while having \mathbf{r}_i look indistinguishable from any $|\mathbf{k}|$ many bits number that is generated from a TRNG.

B Examples in C

The following is a Gasaq key creation example in C. *This is an example to aid in explaining Algorithm 1; not fit for production.*

```
void ghasaq_make_key(
    char *p,      /* password input      */
    FILE *trng,  /* TRNG stream input   */
    FILE *ret,   /* retrieval bits output */
    uint64_t *k, /* 128-bit key output  */
) {
    k[0] = 0;
    k[1] = 0;
    int i, j;
    uint64_t r[2];
    for (i = 0; p[i] != '\n'; i++) {
        for (i = 1; i <= 128; i <<= 1) {
            fread(r, 16, 1, trng);
            k[0] ^= r[0];
            k[1] ^= r[1];
            if (p[i] & i) {
                r[0] ^= 0xffffffffffffffff;
                r[1] ^= 0xffffffffffffffff;
            }
            fwrite(r, 16, 1, ret);
        }
    }
}
```

The following is a key retrieval example in C. *This is an example to aid in explaining Algorithm 2; not fit for production.*

```
void ghasaq_get_key(
    char *p,      /* password input      */
    FILE *ret,   /* retrieval bits input */
    uint64_t *k, /* 128-bit key output  */
) {
    k[0] = 0;
    k[1] = 0;
    int i, j;
    uint64_t r[2];
    for (i = 0; p[i] != '\n'; i++) {
        for (i = 1; i <= 128; i <<= 1) {
            fread(r, 16, 1, ret);
            if (p[i] & i) {
                r[0] ^= 0xffffffffffffffff;
                r[1] ^= 0xffffffffffffffff;
            }
            k[0] ^= r[0];
            k[1] ^= r[1];
        }
    }
}
```