Appendix 13
New Encryption Methods for QKD and RKD

For data encryption using a one-time pad (see Chapter 6 of the book), which is provably 100% secure, the key bits must come from a non-deterministic random number generator (TRNG)—that is, they must be completely random—and the random key bits must be at least as long as the data itself. A major problem here is the required key size—that is, the required number of key bits—which in practice can quickly reach the order of Gigabytes. These enormous key lengths must be generated and distributed using non-deterministic random number generators or, in the case of RKD and QKD, through entanglement inherent to the system, where QKD and RKD quickly reach their limits.

Therefore, it is advantageous if these key lengths can be limited, although this necessarily entails minor restrictions on the one-time pad. This chapter presents four such encryption methods/modes, which were scientifically developed by the book's lead author. These methods/modes were specifically developed for the technologies QKD (Quantum Key Distribution) and RKD (Radio-signal Key Distribution), because QKD and RKD already use the XOR operation and hash functions themselves and therefore do not require any additional mathematical methods for data encryption.

The One-Time method is provably absolutely secure. Its 100% security can be proven under these three conditions:

1. The key must be completely random, i.e., generated by a non-deterministic random number generator.
2. The key must be at least as long as the unencrypted data. If it is longer than the unencrypted data, only a portion of the key is used .
3. New key bits must be used for each new encryption; that is, the key—or any part of it—must never be reused.

The one-time pad can be applied to all five physical technologies described in the book, provided that a sufficiently large amount of key material is available. In

the event that this amount of key material is not available—which can happen quickly with QKD and RKD—these four new encryption methods (modes) are ideal for use. They are presented below.

1 General OTPH and OTPS Encryption Methods

The book describes both encryption methods.

The OTPH (One-Time Pad with Hash Key Derivation Function) and OTPS (One-Time Pad with SHAKE256) encryption methods extend the One-Time Pad with a cyclic buffer of length "p" and HMAC/hash functions. This significantly increases the key length, i.e., the number of key bits.

In the OTPH/OTPS methods, the non-deterministic "Key" is treated as a cyclic buffer, and the length of "Key" must be a prime number "p."

Procedure:

1. First, it is determined which encryption method (OTPH or OTPS) will be used, what the length "a" is for each read operation from the cyclic buffer (which is partly determined by the encryption method itself), and what the size of the cyclic buffer "p" is. In addition, it must be determined how many bits are generated after a read operation from the cyclic buffer for block encryption. That is, how many "T_i" are generated in the HKDF or what the output size is in SHAKE256. **These parameters are called "a" (input length), "p" (length of the cyclic buffer), and "ol" (output length), where "p" must be a prime number.** This means that "a" represents the length of the input to the HKDF/SHAKE256 algorithm, and "ol" represents the length of the output of the HKDF or SHAKE256, respectively. This output is used to encrypt the bits of the data blocks $m_j$using the XOR operation. **Another important parameter is "ec," the encryption counter.** It starts at "ec = 1" and is incremented by one after each read operation of "a" bits from the cyclic buffer. In the book, this read operation of "a" bits is called a round; that is, one read operation from the cyclic buffer occurs per round (note that a round here has nothing to do with a round in the cyclic buffer, which depends on the value "d").
2. The data "m" to be encrypted is divided into data blocks; in practice, these are usually 256-bit or 512-bit blocks. The block length is referred to below as "bl," and the block number as "j." The last block may also be shorter, meaning no padding is required.
3. The "Key" is transferred to the cyclic buffer, and with it, possibly also the values of all "d_i". Otherwise, all "d_i" are stored separately. Then the read

operations begin with a bit spacing (distance between two bits) of "d_1" from the cyclic buffer. "a" bits are always read and fed into the next process, which generates the output—the actual key bits—for data encryption using the XOR operation.

4. Encryption is performed one data block at a time. For each data block, "bl" bits are sequentially extracted from the result of the HKDF or SHAKE256—that is, the output of length "ol." Once all bits from the output have been used up—which occurs after "ol / bl" data blocks—the "ec" (encryption counter) must be incremented by one, and "a" bits must be read from the cyclic buffer and processed, which in turn yields new key bits of length "ol." After at most "p" read operations, the bit spacing, denoted by "d_i", is also changed; that is, "d_i" becomes "d_{i+1}" (e.g., "d_3" becomes "d_4"). This change in "d_i" prevents the repetition of identical read operations. If a second value is specified for each "d_i", this value indicates after how many read operations the "d_i" must be changed.

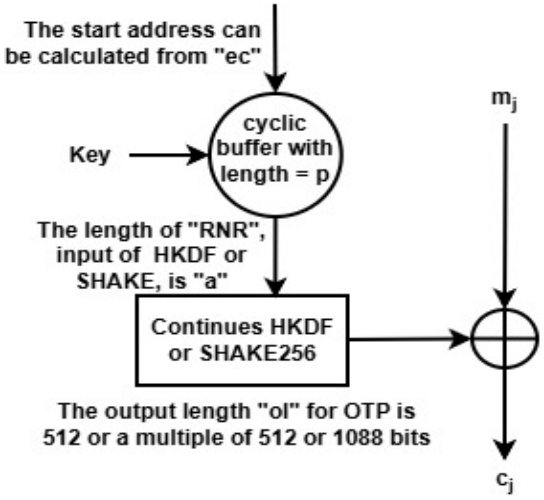


Figure: OTPH/OTPS Encryption Scheme

Since the maximum "d_i" is "d_{p-1}" and only a maximum of "p" read operations are allowed per "d_i", a maximum of "p * (p-1)" read operations are possible, which corresponds to the value "p^2- p". That is, after "p^2– p" read operations, all possible "d_i" have been used, and at that point at the latest, the process must be stopped and a new "Key" must be used.

When a new key for data encryption is transferred to the cyclic buffer, "ec" is set to one. Throughout the entire processing of a "Key" in the cyclic buffer, the variables "a," "p," and "ol" must remain constant, and the values of "d_i" are derived from the "Key." When the key changes—i.e., when new key bits are loaded into the cyclic buffer—these variables may be reset.

For each new read operation of "a" bits from the cyclic buffer, the starting address from which the cyclic buffer must be read is important. Using the new "ec," "a," d_x and "p," the values

- $bp^1, bp^2, bp^3, \ldots\ldots, bp^{a-1}, bp^a$

can be calculated using the following formulas, which specify the individual bit positions in the cyclic buffer for the required read operation of the next "a" bits.

- $bp1 = (ec - 1) * dx * a \bmod p$ **where x = the smallest integer ≥** (ec / p)**,**
- $bp^{i+1} = bp^i + dx \bmod p$ **for all** i **where** $1 < i \leq a$

When decrypting the data, the same "bp" must be used as in the encryption process; that is, in the data storage application, the values of "Key," "a," "p," "ol," and "ec" must be stored. In the telecommunications application, both parties (or more, in the case of multiple telecommunications partners) must use the same "bp" (see Chapter 7 in the book), i.e., they must use the same values for "Key," "a," "p," "ol," and "ec." All possible values of d_x, i.e., d_1, d_2, d_3, …, d_{p-1} are part of "Key," regardless of whether they are at the beginning of the cyclic buffer or separate from it (see the book). The "Key" is changed when the key generation is changed (Key_G), e.g., when an authorized user leaves a role, or when a different role is involved (each role has its own key with its own generations), or when new key material for a key is available in the telecommunications application.

The book assumes that the following always holds: "$d_1 = 1$," and "p" read operations are performed for each d_i. The OTPH and OTPS encryption methods are further generalized in this appendix of the book by treating all $d_i$as three-byte fields located either at the beginning of the "Key" and thus also at the beginning of the cyclic buffer, or completely separate from the cyclic buffer (while still being part of the "Key"). This means that $d_1$is also a random number and not a constant one. And the number of read operations to be performed for each $d_i$is specified in addition to each d_i.

Since the "Key" is extended per round from "a" bytes (input length) to "ol" bytes (output length), and "$p^2 - p$" rounds are possible before a repetition occurs, the length of the key used for encryption is at most the

- extension factor = $(p^2 - p) * ol / a$

greater than the original "Key."

For example, given a key length, i.e., "p", of 20 MByte—which the physical QKD process can generate within a single day over a line connection—and an

extension by the HKDF or SHAKE256 by approximately twenty times, this results in a maximum of around 8,000 TByte per day of keys for data encryption using the XOR operation, an extremely high value per day.

However, this maximum should not be targeted, because at this maximum, each bit of the "Key" (i.e., the individual bits in the cyclic buffer) is used "b * p" times, even though the order of the bits used differs in all "p^2– p" rounds (read operations). However, if the reuse of individual bits in the cyclic buffer is limited to, say, 1000, the extension factor in the above example drops to approximately $2*10^4$,and the 20 MByte still becomes around 400 GByte.

With this key extension, despite the reuse of individual bits in the cyclic buffer, the following applies:

(1) the bits read from the cyclic buffer represent a non-deterministic (completely random) random number,

(2) the order in which the bits are read during each read operation, which is determined by "d_i", changes after a random number of read operations, in which the read interval between the bits is determined by d_i and the number of read operations with this "d_i " is also specified, and both values likewise represent a non-deterministic random number,

(3) the bits read from the cyclic buffer in accordance with (1) and (2) during each read operation (round) are subjected to further processing that is still of great importance from a security standpoint, in that they serve as the input to the HKDF (Hash Key Derivation Function) or SHAKE256 (a variant of the SHA-3 hash algorithm), and only then is the actual key for data encryption generated using the XOR operation.

The OTPH and OTPS methods do not distinguish between data encryption and decryption. That is, the same method and the same key are used for both data encryption and decryption.

2 XTSO Algorithm

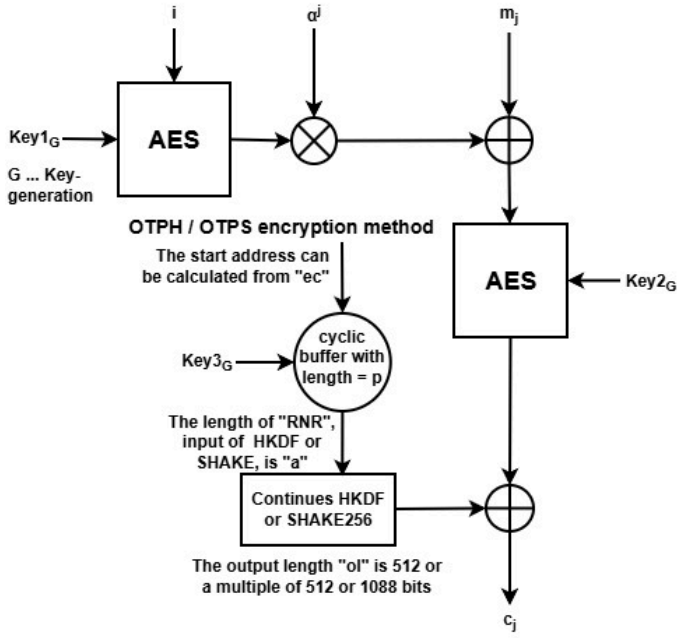
The XTSO algorithm is described in detail in the book and is therefore only briefly covered below. It applies only to data storage.

The XTS algorithm (XEX-based Tweaked CodeBook mode with Ciphertext Stealing) [WP-XTS[1] , see Chapter 6 in the book] is an encryption mode developed for encrypting data on permanent storage devices such as hard drives, SSDs, USB

[1] https://en.wikipedia.org/wiki/Disk_encryption_theory#XTS

flash drives, virtual storage, etc. It was developed by Phillip Rogaway [Rog04[2]] with a few minor modifications. The XTS mode uses two independent keys. XTS was approved by NIST for data storage protection and standardized in 2007 as the quasi-standard IEEE 1619 [WP-IEE[3] , IEEE18[4]].

The new XTSO mode corresponds to the XTS mode (see Chapter 6 in the book), with the following extension: The second XOR function acts as a one-time pad and receives its input from an OTPH or OTPS algorithm.



The XTS and XTSO encryption modes have their own mechanism for the last two data blocks, so that no padding is required. However, this specific mechanism is independent of the XTSO mode and is therefore not discussed here. It can be applied unchanged, as with XTS, to the present XTSO mode as well.

[2] https://doi.org/10.1007/978-3-540-30539-2_2
[3] https://en.wikipedia.org/wiki/IEEE_Security_in_Storage_Working_Group
[4] https://standards.ieee.org/ieee/1619/11552/

3 OTPM Encryption Mode

The OTPM encryption mode (OTPS, OTPH, or OTP with MAC) optionally includes the OTPH encryption algorithm, the OTPS encryption algorithm, or the pure one-time pad and can be used for telecommunications and data storage.

Hereinafter, "OTP" is always used to refer to the encryption method, regardless of whether it is the OTPS method, the OTPH method, or a pure one-time pad. A 512-bit SHA-2 (SHA-512) or SHA-3 is used as the hash function.

In this mode, all unencrypted data—divided into data blocks "m_i" of 512 bits in length—is encrypted into encrypted data blocks, divided into data blocks "c_i" of 512 bits in length. A total of "n" data blocks are encrypted. If the last data block "m_n" is shorter, a correspondingly smaller amount of data is encrypted, which is not a problem with the XOR operation.

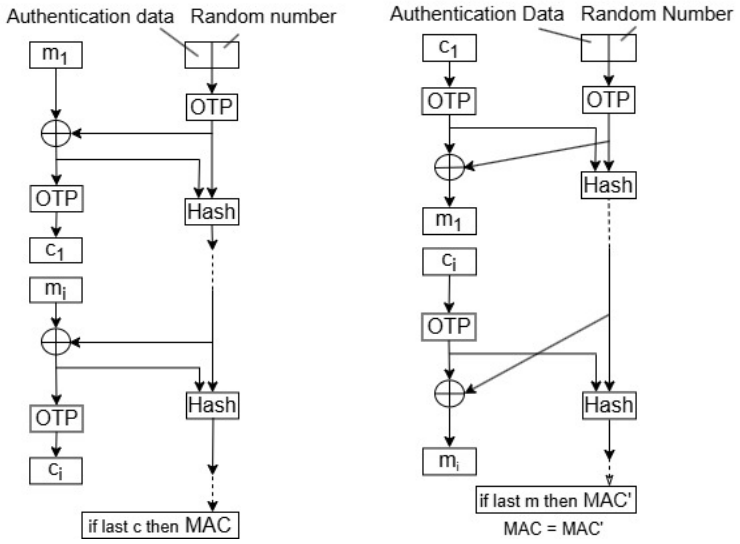


Figure: OTPM mode for decryption and encryption

AD = Authentication Data, RM = Random number

c_1 = OTP (m_1 XOR OTP (AD ‖ RM))

MAC_1 = HASH (m_1 XOR OTP (AD ‖ RM) ‖ OTP (AD ‖ RM))

c_i = OTP (m_i XOR MAC_{i-1})) for all i where $1 < i \leq n$

MAC_i = HASH (m_i XOR MACi-1 ‖ MAC_{i-1}) for all i where $1 < i \leq n$

MAC = MAC_n

Figure: Representation of the mode as a formula for encryption and MAC calculation

In OTPM, in addition to data encryption, a MAC (Message Authentication Code) is also generated, using one of the two new methods (OTPS, OTPH) or the pure one-time pad. This allows this encryption mode to achieve not only confidentiality but also the security objectives of integrity and authenticity.

The OTPM mode consists of two processes that are closely linked. In one of these processes, the data is encrypted in blocks with a block length of 512 bits; in the other process, a MAC is generated. In other words, in addition to data encryption, a MAC is generated in parallel, and the data encryption process is randomized in the process. This means that identical unencrypted data encrypted with the same key will result in different encrypted data and MACs, and this difference arises from a random number used in the first step with or without authentication data. This randomization of the encryption and MAC calculation enhances security.

3.1 Encryption

The mode begins with the encryption of block m_1, followed by the calculation of the MAC using authentication data that can be freely chosen, concatenated with a random number; together, these must be at least 512 bits long, but may also be significantly longer. There is no specified length limit here. The authentication data can also be replaced by a random number or vice versa; in that case, the random number or authentication data must be at least 512 bits long.

During the encryption process, the concatenated authentication data and random number are first encrypted using one of the two encryption methods specified above (OTPH or OTPS) and the value $ec_i = ec_{i-1} + 1$. The result is XORed with the first unencrypted data block m_1, and this result is then encrypted using one of the two methods specified above (OTPH or OTPS). The result yields the first encrypted block c_1. In the MAC process, the encrypted authentication data and random number are concatenated with the result of the XOR operation from the encryption process, and the concatenated result is hashed using a hash algorithm (e.g., SHA3-512). This yields the first result, which is used exclusively in the MAC process.

During the encryption process, for all remaining unencrypted blocks m_2, $m_{(3)}$,m_4, etc., the result of the previous MAC process is first XORed with the current m_i. The result of the XOR operation is then encrypted using one of the two methods specified above (OTPH or OTPS). The result in each case is the encrypted block c_i. In the MAC process, the result of the previous MAC process is concatenated with the result of the XOR operation in the encryption process, and the concatenated result is hashed using a hash algorithm (e.g., SHA3-512). In the MAC process, the hash value yields the next result, which is then used further in the MAC process.

Once all unencrypted data m_i has been encrypted, the final result of the encryption step in the MAC process yields the actual MAC, which is returned as the result by the OTPM algorithm.

3.2 Decryption

The mode begins with the decryption of block c_1, followed by MAC calculation using authentication data concatenated with a random number, which must be identical to those used in the encryption process. Since encryption is performed using an XOR operation, the decryption process is identical; that is, there is no difference between the encryption and decryption processes themselves—differences exist only in the OTPM encryption mode.

During the decryption process, the concatenated authentication data and the random number are first encrypted using one of the two methods specified above (OTPH or OTPS) and the value $c_i = c_{i-1} + 1$. Similarly, c_1 is decrypted using one of the two methods specified above (OTPH or OTPS). The two results are combined using the XOR operation, and the result yields the first unencrypted data block m_1.

In the MAC process, the encrypted authentication data and random number are concatenated with the decrypted result of $c_{(1)}$,and the concatenated result is hashed using a hash algorithm (e.g., SHA3-512). This yields the first result, which is reused in the MAC process.

During the decryption process, for all remaining unencrypted blocks c_2, c_3, c_4, etc., c_i is first decrypted using one of the two methods specified above (OTPH or OTPS). The result is XORed with the current value of the MAC process, and the result of this operation yields the unencrypted data block m_i.

In the MAC process, the current value is concatenated with the decrypted result of c_i and the concatenated result is hashed using a hash function (e.g., SHA3-512). This yields the next result in the MAC process, which is then used in subsequent steps of the MAC process.

Once all encrypted data c_i has been decrypted, the final result of the hashing process in the MAC sequence yields the actual MAC, which must be identical to the MAC generated during data encryption.

3.3 Properties of OTPM Mode

Due to the close link between data encryption and the MAC calculation, the MAC contains, in compressed form, not only the “Authentication Data” and “Random

Number" but also all plaintext data—that is, all m_i. And each new c_i also contains, in compressed form, all previous m_i as well as the "Authentication Data" and "Random Number." However, because a hash function is used, this has no negative impact on past values (i.e., m_i with smaller i). The hash function is also important because on the left side (where encryption takes place), the data blocks are processed using only XOR operations, although OTPH and OTPS themselves also use hash functions.

This also means that, due to the "random number," all c_i are randomized—that is, they depend on this random number.

This close link between data encryption and the MAC calculation, the use of authentication data and a random number, the use of OTPH or OTPS for data encryption, and hash functions with 512-bit output make OTPM a highly secure mode for encrypting and decrypting data and performing MAC calculations, thereby achieving the security objectives of confidentiality, integrity, and authenticity.