WCFB: A Wide Block Encryption for Large Data Sets

Andrey Jivsov

Symantec Corporation, 350 Ellis Street, Mountain View, CA 94043, U.S.A.

Abstract. We define a model for applications that process large data sets in a way that enables additional optimizations of encryption operations. We show how to take advantage of identified characteristics with a new construction of a strong pseudo-random tweakable permutation, WCFB, that is built with $2m + 1$ block cipher invocation for m cipherblocks, plus $\approx 5m$ XOR operations.
WCFB mode has simple structure and is fast in practice. WCFB can benefit from repeated operations on the same wide block and known plaintext.

1 Introduction

Systems that process large data sets often have the following characteristics:

(A) The data set is read and modified in smaller portions, or *blocks*, located at any index in the data set. This applies more to database storage, disk volumes, or cloud storage, as opposed to small files or short messages. An example of a block here is a sector on a disk volume.
(B) On average *multiple blocks* are accessed in one request. This might be because blocks are organized into clusters at a higher level of the system and a cluster is a typical unit that is processed.
(C) While data integrity is important, a common requirement is *no expansion* of the block size, i.e. due to authentication tags. For example, such an expansion property would be unsuitable in many plug-in style application, when the encryption layer is added as an intermediate layer to a system that doesn't support integrity funcionality at that layer.
(D) Encryption of known plaintext is common. For example, consider the encryption of newly allocated blocks, which are filled with zeros. In general, we are trying to take advantage of any *caching* of intermediate results.
(E) We are interested in the performance of the *entire* encryption layer on general-purpose CPUs. We count not only the block-cipher calls, but all other operations including these that simply "mix" data.

Examples of a system that fit the above criteria are database systems, cloud storage, whole disk, volume, or transparent file encryption products; content streaming application, and network proxies. The above characteristics of the system define a *target application*, which performance we seek to optimize here.

In this paper we focus on *tweakable wide block encryption* as a solution to data confidentiality in large data sets.

Jivsov A..
WCFB: A Wide Block Encryption for Large Data Sets.
DOI: 10.5220/0004968900750082
In *Proceedings of the 11th International Workshop on Security in Information Systems* (WOSIS-2014), pages 75-82
ISBN: 978-989-758-031-4

The concrete contribution of this paper is a practical encryption mode **WCFB**, which stands for Wide Cipher FeedBack, that is optimized for the above requiremens. Discussion of practical results calls for further clarification of the *operating environment* in which the implementation of an encryption algorithm is executing. We assume that the operating environment has the following characteristics:

(a) the block cipher (BC) performance is fast
(b) given a generic $\mathsf{GF}(2^n)$ multiplication (GF2mul), the time to perform 2 GF2mul is comparable to 1 BC
(c) XOR is fast.

(See section 2 for notations used above; the reason for comparing BC with GF2mul will be explained in section 3). Our operating environment is justified by the expectation that large data set should be processed on powerful systems with some level of hardware support to make BC fast.

2 Notations

In this paper we consider a new *wide* block cipher, as compared to the (*underlying*) block cipher. The size of the wide block cipher is $l = n * m$ bits. In practical applications the $l/8 \geq 512$ bytes, is a power of two, and is usually a fixed value for a given operating system/hardware/data set. Wide block ciphers work with an underlying block cipher, such as AES-128. One such underlying block cipher call is identified by one BC in section 1. Each wide block P, C is represented by m n-bit blocks, which are denoted as $P_i, C_i : i \in [0, m-1]$. P_i denotes the block of the plaintext such that $P = P_0||P_1||\cdots||P_{m-1}$, with similar definition for the C. P_0 refers to the block that occupies the lowest $n/8$ bytes of the memory range in which P resides. This indexing is known as a little-endian notation.

The WCFB mode is defined with the following types of operations: the underlying block cipher encryption or decryption and $\mathsf{GF}(2^n)$ additions (*XOR* or $\oplus$).

WCFB is a secret key permutation. Besides using underlying block cipher with its key k, it has a set of $m + 1$ derived subkeys k_i, each n bits long regardless of the key k. There is a key schedule for k that the underlying block cipher uses and a set of the subkeys k_i (which is can be viewed as some other key schedule).

By the *data set* we mean a set of logically related wide blocks P, C. The same key k will be used for a given data set. An encrypted storage disk is an example of a data set.

A *tweak* is denoted by T. Typically it is an offset in a data set, or a block index.

Description of loops in iterative algorithms use compact notation defined next. All loops in this paper are based on the default iteration from 0 to $m - 1$ in the ascending direction. The default loop is for $i = 0, 1, \cdots, m - 1$, inclusive; it is noted as $\forall i \curvearrowright$. The same loop in the reverse direction is noted as $\forall i \curvearrowleft$. When the range of indexes is different from the default one, it is always explicitly noted. When the direction is omitted, such as in $\forall i$, it is not important (and so the random order is possible).

We use the following symbols, respectively, for a definition, equality, assignment: $\doteq$, $=$, $\leftarrow$.

3 Comparison with Other Modes

Many wide encryption modes were introduced during the first decade of 2000. Modes with provable security are CMC [1], EME2 [2], PEP [3], TET [4], HEH [5], XCB [6], HCTR [7], HCH [8]. EME2 is standardized in the IEEE 1619.2 section "Wide-Block Encryption" of the IEEE P1619 standard.

There are modes that do not offer security proofs, such as Elephant+CBC [9], modes that do not offer the benefit of a full block permutation such as XTS [10] (XTS is standardized by the IEEE 1619 standard and by NIST), and there are even uses of standard CBC for wide block encryption, motivated by users' dissatisfaction with performance of more secure schemes. Considering the use of wide encryption modes in whole disk encryption products, the overhead of the the crypto code appears as substantial to end-users, especially with now common solid-state storage media. We are not aware of any existing whole disk product that offers a wide block encryption mode.

An overview of current modes is provided in the Table 1 of HEH [5]. Counting two GF2mul as one BC call, the best mode under this accounting is CMC [1] at $2m + 1$ BC operations for a 2-key variant and $2m + 2$ for a 1-key variant. [5] lists HEHfp as $m+1$ BC, $2(m-1)$ GF2mul, which by our accounting is equivalent to $2m$ BC, but it is ignoring other operations that are more complex than XORs. Most importantly, it does not account for additional $2(m-1)$ GF2mul that have one of the multiplication factors random but fixed per the data set. This processing is similiar to EME2's, discussed later in this section. In addition, HEHfp includes $(m-1) \cdot \otimes x$ operations. Finally, it has ≈ 6 XORs per BC.

This brings us back to CMC. Among positives, CMC has $m + 1$ BC and a lean mixing layer (3 data-dependent XORs). On the other hand, CMC has two key schedules (for $m + 1$ BC) and is unable to take any advantage of caching. CMC's first encryption pass is performed in CBC mode with T added at the first step. This denies any benefit of caching of ciphertexts for known plaintext. CMC has $\approx 3m$ XORs for the mixing layer, which is equivalent to WCFB's XORs on modern architectures, because WCFB's $2m$ XORs are data-independent.

Although this may not account for much in practice, two iterations in WCFB are simple back-and-forth pass over the blocks, with the data from an n-bit block used only for the adjacent one, for the ideal CPU cache utilization. CMC performs the mirroring of the block indices between passes.

EME2 mode is close to CMC as a suitable alternative for our operating environment at $2m+1+m/n$ BC. EME2 matches WCFB's caching capability at the first encryption layer. An implementation optimized for bulk performance will use $\approx 3m$ XORs, plus bit operations for m data-dependent GF2mul. One of the factors in these GF2mul is of special form $y(x) = x^i$, so that these m GF2mul can be implemented at a minimum of $2m$ bitwise shifts and m XORs in a sequential manner. While the sequential processing in EME2 contradicts its main design goal, only a small constant-degree parallelism (per CPU core) may practically be realizable and this limited parallelism can be accomplished with reasonable supporting data structures. Overall, WCFB's mixing layer compares well with EME2 and HEHfp: it is a simple XOR sum of n-bit blocks, fully parallelizable.

WCFB is a single-key mode that achieves $2m + 1$ BC and $\approx 5m$ XORs with $2m$ of them data-independent. WCFB has excellent caching capabilities under update scenarios and other repeated access patterns: in the worst case WCFB saves at least one encryption during an update (which corresponds to the encrypted T), while in the best case WCFB can reuse ciphertexts from $m + 1$ encryptions of the n-bit plaintext blocks and T.

WCFB's only operation is XORs on the n-bit blocks. WCFB has no data-dependent lookup, and thus offers an exellent protection against side-channel attacks.

Although this is not critical in the defined operating environment, WCFB and CMC allow full parallelization for 2 out of 3 layers of the encrypt-mix-encrypt processing within each wide block. Despite not achieving clear 3 out of 3 layer full parallelism, EME2 allows parallel execution of either block cipher layers.

The security of WCFB is quadratic in the number of queries, a typical boundary in this category.

Finally, WCFB has the concept of the IV of the data set, which binds the wide block to a respected data set, a unique feature to WCFB.

4 Our Contribution

Our main goal is to make encryption of large data sets faster in practice. Two contributions towards this goal are presented here.

First, we defined the target applications and operating environment in such terms that allow additional optimizations. For example, the wide encryption modes were traditionally valuing the internal parallelism of the mode, i.e. its ability to process multiple n-bit blocks within the nm-wide block, however, section 1 asks if the multi-wide-block parallelism is better concept to exploit the parallelism. We argue that it is the case for many target applications, and this assumption allows simplification of the mode.

Second, we propose the new mode WCFB, which is a "tweakable" mode to allow changes to random "wide" blocks in an encrypted data set. We describe the WCFB in details and highlight its advantages for processing large data sets. Some design ideas employed in WCFB, in particular the ability to cache the ciphertexts or subkeys without any overhead for this option, are general techniques with applications beyond WCFB.

For the operating environment defined in section 1 WCFB is a wide encryption mode that has *operation count* of $\boxed{2m + 1 \text{ BC}}$ with a lean mixing layer at $\boxed{\approx 5m\text{XORs}}$ (see section 3 for the comparison with other modes).

$2m + 1$ operation count is a reasonable threshold for a tweakable wide encryption mode of encrypt-mix-encrypt, given that there is a mixing step, a tweak, and an IV that need to be "processed". We show next how caching lowers the metric to $2m$ or lower BC.

There are two likely events that can be relied on in this respect: a favorable *usage pattern* and *low entropy* (n-bit) plaintexts.

Consider the *update* usage pattern, which we define as subsequent decrypt and encrypt operations on the same nm-bit block, either performed as a unified operation, or close enough in time so that some intermediate results can be efficiently re-used. This

particular order corresponds to a very common access pattern to encrypted data set: reading a random block in an encrypted data set, decrypting it, making modifications to the plaintext, encrypting it, and then writing back at the same position. Further, consider an application that only adds data to a file. While an application adds a byte at a time to a file, at the lower level of the operating system the storage can only be accessed in blocks, given that the storage devices are block devices. If the wide block encryption is employed for the protection of the disk blocks, even consecutive minor file append operations will likely result in update (or write only) operations to the same disk block.

The encryption of low-entropy data is quite common in practice as well. Any fixed-size data set is expected to use a fixed-value padding, typically with zeros. There are many high-level operations that fall into this category, such as zeroization of data set blocks; WCFB will only need m BC to accomplish a zeroization request on a wide block, on par with CBC performance.

WCFB implementations can fully benefit from caching, primarily owing to WCFB's ECB-style first pass of encryption.

5 Specification of WCFB

The algorithm is defined in Fig. 1.

The WCFB follows the encrypt-mix-encrypt approach. It is built from two passes over n-bit blocks that are CFB-like and CBC-like. The mix step corresponds to the XOR of intermediate n-bit values. WCFB can be viewed as having a double nested structure WCFB$[\hat{E}[E]]$, where WCFB is, by and large, defined in terms of $\hat{E}$.

The run-time data-dependent input to WCFB encryption or decryption is a wide block P or C, respectively (steps 5-10), and the corresponding tweak T.

Other input values, κ and IV, are fixed for a given data set; they are pre-processed during initialization, steps 1-4. Additional values may need to be calculated to take advantage of caching features of WCFB.

Initialization Vector

WCFB requires a unique IV per data set for the same κ (κ is the shared secret, defined bellow). The uniqueness v.s. randomness condition on the IV is justified because it is only used as a value $\hat{E}_{k,k_0}(\mathsf{IV})$ in a standard CFB mode with n-bit block size. IV offers an additional method to segregate data sets, in addition to using a different κ. Note that this handling of IV adds robustness in practice because high-quality nounces are not critical for WCFB.

Key Set up at the Step 2

WCFB mode uses a single symmetric key κ. This key is "expanded" into the main key k, and $m+1$ subkeys k_i using a key derivation function $\mathsf{KDF}(\kappa, \mathsf{IV}, i)$, where the parameter i is the index or the returned material. This step is performed once per data set.

The precise details of the KDF are unimportant, as long as $\{k, k_0, \ldots, k_m\}$ are indistinguishable from selected uniformly at random, even when the attacker has access

Fig. 1. WCFB algorithm.

Encryption	Decryption
1: $C_{-1} \doteq \mathsf{IV}$	
2: $k \leftarrow \mathsf{KDF}(\kappa, \mathsf{IV}, 0), \forall_{i=0}^{m} i : k_i \leftarrow \mathsf{KDF}(\kappa, \mathsf{IV}, i+1)$	
3: define $\hat{E}_{k,k_i}(\cdot) \doteq E_k(\cdot \oplus k_i)$, $\hat{E}^{-1}_{k,k_i}(\cdot)$ is its inverse	
4: $P_m \doteq \hat{E}_{k,k_m}(T)$	
5: $\forall i \curvearrowleft$: $P'_i \leftarrow \hat{E}_{k,k_i}(P_i) \oplus P_{i+1}$	$\forall i \curvearrowright$: $P_i \leftarrow \hat{E}_{k,k_i}(C_{i-1}) \oplus C_i$
6: $\forall i : P_i \leftarrow P'_i$	$S \leftarrow \hat{E}_{k,k_m}(\bigoplus_{i=1}^{m-1} P_i)$
7: $P_{m-1} \leftarrow P_{m-1} \oplus P_0$	$P_0 \leftarrow P_0 \oplus S$
8: $S \leftarrow \hat{E}_{k,k_m}(\bigoplus_{i=1}^{m-1} P_i)$	$P_{m-1} \leftarrow P_{m-1} \oplus P_0$
9: $P_0 \leftarrow P_0 \oplus S$	$\forall i \curvearrowleft$: $P_i \leftarrow \hat{E}^{-1}_{k,k_i}(P_i \oplus P_{i+1})$
10: $\forall i \curvearrowright$: $C_i \leftarrow \hat{E}_{k,k_i}(C_{i-1}) \oplus P_i$	

to an oracle providing $E_k(\cdot), E_k^{-1}(\cdot)$. The last clause is necessary because the discovery of one subkey enables an oracle for $E_k(\cdot), E_k^{-1}(\cdot)$, and this may help provide additional information on other subkeys (a trivial case of $k_i = E_k(i)$ is a poor choice for this reason).

We offer flexibility in selecting the KDF to meet external requirements on key derivation. In many cases, such as when the shared secret κ is obtained through a higher-level key exchange protocol, a KDF is already defined. In these cases the KDF is executed more times to get the needed key material.

Alternatively, we could instantiate WCFB as a two-key mode with some κ_0 and κ_1 and avoid the KDF entirely.

Operation Count

There are $2m + 1$ encryptions, plus one encryption of the IV, $P_m = \hat{E}_{k,k_0}(\mathsf{IV})$, which is fixed for the data set, and which lifecycle is identical to the lifecycle of the subkeys k_i. It should be cached along with the subkeys.

The rest of operations are $\approx 5m$ XORs and assignments. Each of $\hat{E}$ includes one XOR, therefore, only $\approx 3m$ XORs are data-dependent, and only m block cipher operations cannot be fully parallelized (as is the case for CBC encryption).

Update scenario, introduced in section 4, allows for the caching of plaintexts/ciphertext pairs that are shared between decryption of some ciphertext to corresponding plaintext P, followed by the encryption of a similar to P plaintext, for the same T. In the worst case, we always can reuse P_m (Fig.1, step 4) that corresponds to (data-independent) T. The best case scenario represents changes to a single n-bit plaintext block. The first step in the encryption direction is identical to CBC decryption, and in this case only one of the m encryptions on line 5 will be for the new plaintext. Thus, m out of $m + 1$ prior n-bit ciphertexts can be reused at the step 5 of encryption in the best case.

Concrete Security

Most of the remaining content of the paper is the proof of (1) in section 6. This upper bound means that in order to distinguish 512-byte WCFB with AES-128 from a random

permutation with probability of 0.5 an attacker must obtain 2^{54} plaintext/ciphertexts pairs, 512 bytes each, assuming that there is no better attack on the AES-128. This is 2×2^{33} TiB, well above today's storage capability.

6 Security of WCFB

Theorem 1. *For any attacker A that can perform up to q queries consisting of mn-bit request/response pairs, it holds that the A's advantage to distinguish a WCFB instantiated with a random PRP operating on a n-bit domain from a random tweakable PRP operating on a nm-bit domain has the following upper bound:*

$$\mathbf{Adv}^{\pm\widetilde{\mathsf{prp}}}_{\mathit{WCFB}[\mathsf{Perm}(n)]}(q) < 1.5q^2(m+1)^2 2^{-n} \tag{1}$$

The theorem means that when we instantiate WCFB with an ideal primitive modeling a block cipher, we get the insecurity directly attributed to WCFB as specified in (1).

Other related "advantage notions" can be obtained from (1) by plugging (1) into inequalities that were proven to hold for wide encryption modes in general. For example, if WCFB is instantiated with a block cipher, we use results from (1) as follows:

$$\mathbf{Adv}^{\pm\widetilde{\mathsf{prp}}}_{\mathsf{WCFB}[E]}(t,q) < \mathbf{Adv}^{\pm\widetilde{\mathsf{prp}}}_{\mathsf{WCFB}[\mathsf{Perm}(n)]}(q) + 2\mathbf{Adv}^{\pm\mathsf{prp}}_{E}(t',q) \tag{2}$$

(2) comes from the [1], where we refer the reader in the interest of saving space.

The result of theorem 1 is that WCFB provides birthday-bound security in terms of the underlying block cipher, typical boundary for wide encryption modes. The proof is constructed by showing that inner operations of the WCFB behave as PRF when the number of queries q is bound, provided that the m-bit block cipher is modeled as a PRF. Under these bounds the insecurity is the probablilty to distinguish inner PRFs from a random function. The inner probalbilities are combined with the distinquishing advantage of PRP v.s. PRF to arrive at (1). The proof is provided in the full version of this paper.

References

1. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In Boneh, D., ed.: Advances in Cryptology - CRYPTO 2003. Volume 2729 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2003) 482–499
2. Halevi, S.: Eme*: Extending eme to handle arbitrary-length messages with associated data. In Canteaut, A., Viswanathan, K., eds.: Progress in Cryptology - INDOCRYPT 2004. Volume 3348 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 315–327
3. Chakraborty, D., Sarkar, P.: A new mode of encryption providing a tweakable strong pseudo-random permutation. In Robshaw, M., ed.: Fast Software Encryption. Volume 4047 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2006) 293–309
4. Halevi, S.: Invertible universal hashing and the tet encryption mode. In Menezes, A., ed.: Advances in Cryptology - CRYPTO 2007. Volume 4622 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2007) 412–429

5. Sarkar, P.: Improving upon the tet mode of operation. In Nam, K.H., Rhee, G., eds.: Information Security and Cryptology - ICISC 2007. Volume 4817 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2007) 180–192
6. McGrew, D., Fluhrer, S.: The security of the extended codebook (xcb) mode of operation. In Adams, C., Miri, A., Wiener, M., eds.: Selected Areas in Cryptography. Volume 4876 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2007) 311–327
7. Wang, P., Feng, D., Wu, W.: Hctr: A variable-input-length enciphering mode. In Feng, D., Lin, D., Yung, M., eds.: Information Security and Cryptology. Volume 3822 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 175–188
8. Chakraborty, D., Sarkar, P.: Hch: A new tweakable enciphering scheme using the hash-encrypt-hash approach. In Barua, R., Lange, T., eds.: Progress in Cryptology - INDOCRYPT 2006. Volume 4329 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2006) 287–302
9. Ferguson, N.: Aes-cbc + elephant diffuser: A disk encryption algorithm for windows vista (2006)
10. Martin, L.: Xts: A mode of aes for encrypting hard disks. Security Privacy, IEEE **8** (2010) 68–69