The Extended Codebook (XCB) Mode of Operation

David A. McGrew and Scott Fluhrer
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95032
{mcgrew,sfluhrer}@cisco.com

October 25, 2004

Abstract

We describe a block cipher mode of operation that implements a 'tweakable' (super) pseudorandom permutation with an arbitrary block length. This mode can be used to provide the best possible security in systems that cannot allow data expansion, such as disk-block encryption and some network protocols. The mode accepts an additional input, which can be used to protect against attacks that manipulate the ciphertext by rearranging the ciphertext blocks.

Our mode is similar to a five-round Luby-Rackoff cipher in which the first and last rounds do not use the conventional Feistel structure, but instead use a single block cipher invocation. The third round is a Feistel structure using counter mode as a PRF. The second and fourth rounds are Feistel structures using a universal hash function; we re-use the polynomial hash over a binary field defined in the Galois/Counter Mode (GCM) of operation for block ciphers. This choice provides efficiency in both hardware and software and allows for re-use of implementation effort. XCB also has several useful properties: it accepts arbitrarily-sized plaintexts and associated data, including any plaintexts with lengths that are no smaller than the width of the block cipher.

This document is a pre-publication draft manuscript.

Contents

1 Introduction

There are several scenarios in which *length-preserving, deterministic encryption* is useful. An encryption method is length-preserving if the ciphertext has exactly the same number of bits as does the plaintext. Such a method must be deterministic, since it is impossible to accommodate random data (such as an initialization vector) within the ciphertext. In some cases, deterministic length-preserving encryption exactly matches the requirements. For example, in some encrypted database applications, determinism is essential in order to ensure a direct correspondence between plaintext values being looked up and previously stored ciphertext values.

In some other cases, there is a length-preservation requirement that makes it impossible to provide all of the security services that are desired. Length-preserving algorithms cannot provide message authentication, since there is no room for a message authentication code, and they cannot meet strong definitions of confidentiality [2]. Essentially, these algorithms implement a codebook; repeated encryptions of the same plaintext value with the same key result it identical ciphertext values. An adversary gains knowledge about the plaintext by seeing which ciphertext values match, and which do not match. Despite these limitations, in many scenarios it may be desirable to use length-preserving encryption because other methods are unworkable. Length-preservation may allow encryption to be introduced into data processing systems that have already been implemented and deployed. Many network protocols have fixed-width fields, and many network systems have hard limits on the amount of data expansion that is possible. One important example is that of disk-block encryption, which is currently being addressed in the IEEE Security in Storage Working Group [5].

Given the limitation of length-preservation, the best security that we can provide is *non-malleable* encryption. Informally, a cipher is non-malleable if changing a single bit of a ciphertext value affects all of the bits of the corresponding plaintext. More formally, we require our cipher to be a *pseudorandom permutation*; it is indistinguishable from a permutation on the set of messages to a computationally bounded adversary. Because we want our cipher to handle plaintexts whose size may vary, we require the cipher to be a pseudorandom *arbitrary length* permutation: for each of the possible plaintext lengths, the cipher acts as a pseudorandom permutation. To provide as much flexibility as possible, we allow the plaintext lengths to vary even for a single fixed key.

In some cases, some additional data can be associated with the plaintext. By using this data as an input, we can provide better security, by letting each distinct associated data value 'index' a pseudorandom permutation. That is, we require the cipher to be a *pseudorandom arbitrary-length permutation with associated data*: for each plaintext length and each value of the associated data, the cipher acts as a pseudorandom permutation. For maximum flexibility, we allow the length of the associated data field to vary even for a single fixed key. In the disk block example, we can use the block number as the associated data value. This will prevent some attacks which rely on the codebook property, since identical plaintext values encrypted with distinct associated data values give unrelated ciphertext values.

The use of an associated data input to a pseudorandom permutation first appeared in the innovative Hasty Pudding Cipher of Schroppel [10], where it was called a 'spice', and was given a rigorous mathematical treatment by Liskov, Rivest, and Wagner [7], who called it a 'tweak'. Our

security goal follows that of the latter work, with the distinction that we allow the associated data to have an arbitrary length.

The extended codebook (XCB) mode of operation for block ciphers implements a pseudorandom arbitrary-length permutation with associated data. XCB is defined so that it can use any block cipher, but we provide test data only for AES-128. This mode is amenable to implementation in both hardware and software, and it has a computational cost that is relatively low (compared to similar modes): it only requires $n + 1$ block cipher invocations and $2n$ multiplications in $GF(2^w)$, where w is the number of bits in the block cipher inputs and outputs. The mode also has several useful properties: it accepts arbitrarily-sized plaintexts and associated data, including any plaintexts with lengths of at least w bits. This property allows XCB to protect short data, like the common 20-byte G.729 voice codec in Secure RTP [1].

There are several other block cipher modes of operation that also implement psedurandom arbitrary-length permutations. Most notable is the EME mode of Halevi and Rogaway [4], which is also efficient and accepts associated data.

In the following, we define XCB (Section 2) and analyze its security (Section 3). Test data for XCB mode for the Advanced Encryption Standard with 128 bit keys (AES-128) [9] is provided in Appendix A.

2 Specification

This section contains the complete normative specification for XCB for use with 128-bit block ciphers. In order to use XCB with other block cipher widths, it is necessary to define a finite field of the appropriate size.

2.1 Notation

The two main functions used in XCB are block cipher encryption and multiplication over the field $GF(2^{128})$. The block cipher encryption of the value X with the key K is denoted as $\mathbf{e}(K, X)$, and the block cipher decryption is denoted as $\mathbf{d}(K, X)$. (Note that we reserve the symbols E and D to denote XCB encryption and decryption, respectively.) The number of bits in the inputs and outs of the block cipher is denoted as w. For the Advanced Encryption Standard (AES), $w = 128$. The multiplication of two elements $X, Y \in GF(2^{128})$ is denoted as $X \cdot Y$, and the addition of X and Y is denoted as $X \oplus Y$. Addition in this field is equivalent to the bitwise exclusive-or operation, and the multiplication operation is defined in Section 2.3.

The function $\mathrm{len}(S)$ returns a 64-bit string containing the nonnegative integer describing the number of bits in its argument S, with the least significant bit on the right. The expression 0^l denotes a string of l zero bits, and $A \| B$ denotes the concatenation of two bit strings A and B. We consider bit strings to be indexed starting on the left, so that bit zero of S is the leftmost bit. When S is a bit string and $0 \le a < b \le \mathrm{len}(S)$, we denote as $S[a; b]$ the length $b - a$ subtring of S consisting of bits

a through b of S. The symbol $\{\}$ denotes the bit string with zero length.

2.2 Definition

The XCB encryption and decryption operations are defined in Algorithms 1 and 2, respectively, and he encryption operation is illustrated in Figure 1. These algorithms use the block cipher encryption functions $\mathbf{e}$ and $\mathbf{d}$, as well as the hash function $\mathbf{h}$ and the pseudorandom function $\mathbf{c}$. The round keys K_0, K_1, K_2, K_3, K_4 could be stored between evaluations of these algorithms[1].

Algorithm 1 The XCB encryption operation. Given a key $K \in \{0,1\}^k$, a plaintext $\mathbf{P} \in \{0,1\}^m$ where $m \in [w, 2^{39}]$, and associated data $\mathbf{Z} \in \{0,1\}^n$ where $n \in [0, 2^{39}]$, returns a ciphertext $\mathbf{C} \in \{0,1\}^m$.

$K_0 \leftarrow \mathbf{e}(K, 0^w)$, $K_1 \leftarrow \mathbf{e}(K, 0^{w-1}\|1)$, $K_2 \leftarrow \mathbf{e}(K, 0^{w-2}\|1\|0)$,
  $K_3 \leftarrow \mathbf{e}(K, 0^{w-2}\|1^2)$, $K_4 \leftarrow \mathbf{e}(K, 0^{w-3}\|1\|0^2)$
$A \leftarrow \mathbf{P}[0; w-1]$
$B \leftarrow \mathbf{P}[w; \mathsf{len}(\mathbf{P}) - 1]$
$C \leftarrow \mathbf{e}(K_0, A)$
$D \leftarrow C \oplus \mathbf{h}(K_1, B, \mathbf{Z})$
$E \leftarrow B \oplus \mathbf{c}(K_2, D)$
$F \leftarrow D \oplus \mathbf{h}(K_3, E, Z)$
$G \leftarrow \mathbf{d}(K_4, F)$
return $G\|E$

Algorithm 2 The XCB decryption operation. Given a key $K \in \{0,1\}^k$, a ciphertext $\mathbf{C} \in \{0,1\}^m$ where $m \in [w, 2^{39}]$, and associated data $\mathbf{Z} \in \{0,1\}^n$ where $n \in [0, 2^{39}]$, returns a plaintext $\mathbf{P} \in \{0,1\}^m$.

$K_0 \leftarrow \mathbf{e}(K, 0^w)$, $K_1 \leftarrow \mathbf{e}(K, 0^{w-1}\|1)$, $K_2 \leftarrow \mathbf{e}(K, 0^{w-2}\|1\|0)$,
  $K_3 \leftarrow \mathbf{e}(K, 0^{w-2}\|1^2)$, $K_4 \leftarrow \mathbf{e}(K, 0^{w-3}\|1\|0^2)$
$G \leftarrow \mathbf{C}[0; w-1]$
$E \leftarrow \mathbf{C}[w; \mathsf{len}(\mathbf{P}) - 1]$
$F \leftarrow \mathbf{e}(K_4, G)$
$D \leftarrow F \oplus \mathbf{h}(K_3, E, \mathbf{Z})$
$B \leftarrow E \oplus \mathbf{c}(K_2, D)$
$C \leftarrow D \oplus \mathbf{h}(K_1, B, Z)$
$A \leftarrow \mathbf{d}(K_0, C)$
return $A\|B$

The function $\mathbf{c} : \{0,1\}^k \times \{0,1\}^w \rightarrow \{0,1\}^{0:2^{39}}$ generates an arbitrary-length output by running

[1]The use of distinct keys in each round is a burden, because those values must either stored in memory and fetched when needed, or computed from the master key when needed. We believe that it is possible to re-use key material between rounds, but have not yet thoroughly analyzed the security of those cases. We chose the conservative security option for the initial version of XCB, but we hope to offer a reduced number of round keys in a future version.

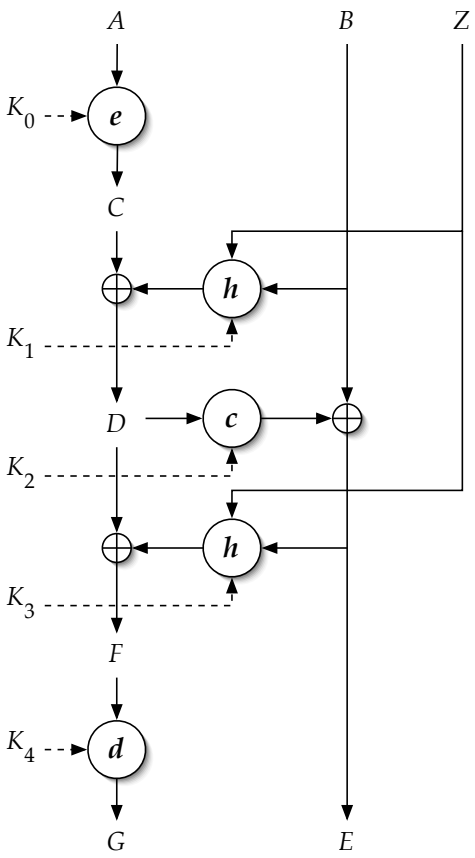

Figure 1: The XCB encryption operation.

the block cipher **e** in counter mode, using its input as the initial counter value. Its definition is

$$\mathbf{c}(K, W) = E(K, W) \| E(K, \text{incr}(W) \| \ldots \| \text{MSB}_t(E(K, \text{incr}^{n-1}(W)), \tag{1}$$

where $n = \lceil l/w \rceil$ is the number of blocks in the output and $t = l \bmod w$ is number of bits in the trailing block. Here incr : $\{0,1\}^w \to \{0,1\}^w$ is the increment function that is used to generate successive counter values. This function treats the rightmost 32 bits of its argument as a nonnegative integer with the least significant bit on the right, increments this value modulo 2^{32}. More formally,

$$\text{incr}(X) = X[0; w-33] \,\|\, (X[w-32; w-1] + 1 \bmod 2^{32}), \tag{2}$$

where we rely on the implicit conversion of bit strings to integers.

The function $\mathbf{h} : \{0,1\}^w \times \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^w, m \in [w, 2^{39}], n \in [0, 2^{39}]$ is defined by

$\mathbf{h}(H, A, C) = X_{m+n+1}$, where the variables $X_i \in \{0,1\}^w$ for $i = 0, \ldots, m+n+1$ are defined as

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & \text{for } i = 1, \ldots, m-1 \\ (X_{m-1} \oplus (A_m^* \| 0^{w-v})) \cdot H & \text{for } i = m \\ (X_{i-1} \oplus C_{i-m}) \cdot H & \text{for } i = m+1, \ldots, m+n-1 \\ (X_{m+n-1} \oplus (C_n^* \| 0^{w-u})) \cdot H & \text{for } i = m+n \\ (X_{m+n} \oplus (\text{len}(A) \| \text{len}(C))) \cdot H & \text{for } i = m+n+1. \end{cases} \tag{3}$$

Here we let A_i denote the w-bit substring $A[(i-1)w; iw-1]$, and let C_i denote $C[(i-1)w; iw-i]$. In other words, A_i and C_i are the i^{th} blocks of A and C, respectively, if those bit strings are decomposed into w-bit blocks. This function is identical to GHASH, the universal hash that is used as a component of the Galois/Counter Mode (GCM) of Operation [8], except that GHASH requires $w = 128$, as is the case for AES [9].

2.3 Multiplication in $GF(2^{128})$

The multiplication operation is defined as an operation on bit vectors in order to simplify the specification; it allows us to keep finite field mathematics out of the normative definition of the algorithm. Background information on this field and its representation, and strategies for efficient implementation, is provided in the GCM specification [8, Sections 3 and 4]. This definition of multiplication corresponds to the polynomial basis with the field polynomial of $f = 1 + \alpha + \alpha^2 + \alpha^7 + \alpha^{128}$.

Each field element is a vector of 128 bits. The i^{th} bit of an element X is denoted as X_i. The leftmost bit is X_0, and the rightmost bit is X_{127}. The multiplication operation uses the special element $R = 11100001 \| 0^{120}$, and is defined in Algorithm 3. The function rightshift() moves the bits of its

Algorithm 3 Multiplication in $GF(2^{128})$. Returns $Z = X \cdot Y$, where $X, Y, Z \in GF(2^{128})$.

```
Z ← 0, V ← X
for i = 0 to 127 do
  if Y_i = 1 then
    Z ← Z ⊕ V
  end if
  if V_127 = 0 then
    V ← rightshift(V)
  else
    V ← rightshift(V) ⊕ R
  end if
end for
return Z
```

argument one bit to the right. More formally, whenever $W = \text{rightshift}(V)$, then $W_i = V_{i-1}$ for $1 \le i \le 127$ and $W_0 = 0$.

3 Security

We provide a proof sketch for the security of XCB in the concrete security model introduced by Bellare et. al. [3]. We can show that XCB is a secure pseudorandom arbitrary-length permutation with associated data (ALPA), using only the assumption that $\mathbf{e}$ is a secure w-bit pseudorandom permutation, as follows. We start by showing that a pseudorandom arbitrary-length function with associated data (ALFA) is nearly indistinguishable from an arbitrary-length PRP with associated data, as long as the number of invocations of the function is bounded. We then show that XCB is nearly indistinguishable from a pseudorandom ALFA, as long as the number of queries to the oracle representing $\mathbf{e}$ is bounded. Tying these two facts together shows that XCB is a pseudorandom ALPA when these bounds are respected. The number of invocations of the XCB encrypt and decrypt functions is less than the number of block cipher encryptions and decryptions used during those invocations. This fact ensures that the security degradation due to our viewing XCB as a PRF is small.

We next sketch how to show that XCB is a secure pseudorandom ALFA against either adaptive chosen plaintext attacks or adaptive chosen ciphertext attacks. (We defer consideration of attacks in which an adversary is allowed to adaptively choose *both* plaintexts and ciphertexts until later.) The basic idea behind this proof is that the ciphertext value $\mathbf{d}(K_4, F)\|(B \oplus \mathbf{c}(K_2, D))$ returned from an encryption query is indistinguishable from random as long as the values of the variables D and F do not repeat across different invocations of that function, and the functions $\mathbf{c}$ and $\mathbf{e}$ are indistinguishable from random. Similarly, the plaintext values $\mathbf{e}(K_0, C)\|(E \oplus \mathbf{c}(K_2, D))$ returned from a decryption query are indistinguishable from random as long as the values of D and C do not repeat. The proof that counter mode is secure is standard [2], as is the effectiveness of a block cipher as a pseudorandom function. The probable uniqueness of the variables C, D and F across all invocations follows from the properties of the function $\mathbf{h}$, which is ϵ-almost xor universal [6]. A detailed treatment requires consideration of the probability that no collision on those variables occurs on the i^{th} query, given that no collision occurred on any of the previous $i-1$ queries.

Security against attacks in which the adversary can adaptively choose both ciphertexts and plaintexts can be proven using a method similar to that outlined above. We give the adversary access to ALFA encryption and decryption oracles, and allow her q queries in total. We assume without loss of generality that the adversary never repeats a query, and never asks for the decryption of a ciphertext value returned by a previous encryption query, and never asks for the encryption of a plaintext value returned by a previous decryption query.

As above, we rely on the absence of collisions for the variables C, D, and F, but in this case we need to more carefully define what a collision means. More precisely, we can show that, if an event Γ occurs, then XCB cannot be effectively distinguished from a random ALFA. This event is defined as the conjunction $\Gamma = \Gamma_0 \cap \Gamma_1 \cap \Gamma_2 \cap \Gamma_3$ of the events defined as follows:

Γ_0 is the event that both $D_i \neq D_j$ and $F_i \neq F_j$ for each pair of distinct encryption queries (i, j).

Γ_1 is the event that both $D_i \neq D_j$ and $C_i \neq C_i$ for each pair of distinct decryption queries (i, j).

Γ_2 is the event that both $D_i \neq D_j$ and $F_i \neq F_j$ for each pair (i, j) of queries consisting of an encryption query i and a decryption query j, where $j < i$.

Γ_3 is the event that both $D_i \neq D_j$ and $C_i \neq C_j$ for each pair (i, j) of queries consisting of an encryption query i and a decryption query j, where $j > i$.

Events Γ_1 and Γ_2 correspond to security against chosen plaintext and chosen ciphertext attacks, respectively. Event Γ_3 takes into account that an adversary can attempt to force an encryption query to cause a collision on D or F with a previous decryption query. Event Γ_4 takes into account that an adversary can attempt to force a decryption query to cause a collision on C or D with a previous encryption query. Of course, an adversary can easily cause a collision on the C-values with two encryption queries that use the same A-values, but this fact is irrelevant because it does not lead to a method of distinguishing XCB from a random ALFA. Similarly, collisions on F-values from two decryption queries are possible but irrelevant.

References

[1] M. Baugher , D. McGrew, M. Naslund, E. Carrara, K. Norrman. "The Secure Real-time Transport Protocol," *IETF RFC 3711*, March 2004.

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. "A concrete security treatment of symmetric encryption," In *Proceedings of the 38th FOCS*, IEEE Computer Society Press, 1997.

[3] M. Bellare, J. Kilian, P. Rogaway, "The Security of the Cipher Block Chaining Message Authentication Code," *J. Comput. Syst. Sci.* 61(3). pg. 362-399 (2000).

[4] S. Halevi and P. Rogaway, A Parallelizable Enciphering Mode, *2004 RSA Conference Cryptography Track*, LNCS, Springer-Verlag, 2004.

[5] IEEE Security in Storage Working Group, Web page. http://siswg.org.

[6] H. Krawczyk, "LFSR-based hashing and authentication," In Y. Desmedt, editor, *CRYPTO '94*, LNCS, Springer-Verlag, Aug. 1994.

[7] M. Liskov, R. Rivest and D. Wagner. Tweakable Block Ciphers. *CRYPTO '02*, LNCS, Springer-Verlag, 2002.

[8] D. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," *Submission to NIST Modes of Operation Process*, January, 2004. Available online at http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes.

[9] U.S. National Institute of Standards and Technology. The Advanced Encryption Standard. *Federal Information Processing Standard* (FIPS) 197, 2002.

[10] R. Schroeppel, *Hasty Pudding Cipher Specification*, First AES Candidate Workshop, August, 1998. Available online at http://www.cs.arizona.edu/people/rcs/hpc/hpc-spec.

A Test Data

In this section we provide detailed test data for AES-128 XCB. The variables are as defined above. The variable X_i corresponds to the invocation of $\mathbf{h}$ that uses the key K_1, while X'_i corresponds to the invocation with the key K_3. The variable Y_i denotes the i^{th} counter block, that is, the value $\text{incr}^{i-1}(W)$ from Equation 1. The variable C_i denotes $\mathbf{e}(K_2, W_i)$. All values are in hexadecimal, and values that are larger than 128 bits in length are continued on the following lines.

All data correspond to a single test case in which the plaintext $\mathbf{P}$ is 512 bytes long and the associated data $\mathbf{Z}$ is 16 bytes long.

Input	Value
K	000102030405060708090a0b0c0d0e0f
$\mathbf{P}$	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000

Input	Value
C	55d00a317ddff00d731e03cdadaa6d81 0d683b183a3e178ced28c7340175c7b6 7611cf25eb0e9a2318b798c643d9815c 723422504806dcf4d7892ca280d9c63b b4230e84ff29ef575b525b82fabe023e 59cb9441d39a146aeee0c41bee6f51ea dceba56600a6c503a611a201543758f1 2553571570f61b93a5e88f6044e8b49d 854aca2c455cf37a26bb56a81b736998 fca69195940476076bc4fdbedfa55cb8 46632a0fd4ab01b29a8e40519ffc476b e992de544e435b33c8664e5e05281a0c 8ce8ffeb1f54d9cd13a4523caace820d c07cc362b234989295f1384dedad9a37 4ebe9cb3b9bb68f6bca463b13d2f128b 3548cd1b1683e2aba097cc288dfe3bd1 4b4b272ac5517bb17b86e52862ae8df4 d5e7683acee2bf39818d6a774935ae0e 609cc97a21174b9f8543d2836b813ba6 3d6c9778a076c71613a2da9cf857f73a b02558a3f064861e7872c2889167cc78 d219f41fae7b511ea2b00aa219101678 6573424b1aa608e4188e053d708e5fc9 22462845bf932ef4995f25905fbfbace 79ff8398153e287f991946786ec33d38 bd5535a3f9be795ec9536907b76c0885 2554a62b304dcbc83ea60ac13d7571fd 15414a7d2bb6770b86434bc779f08285 f2a16a17d43a844b2e607ec26eb4e4e2 3b6547782ea97975315a0c835f2f8b99 49ffa58d04d8c7e218609e817f737a8d 14a24a064811e14ad5eef66714a71e6c

Variable	Value
K_0	c6a13b37878f5b826f4f8162a1c8d879
K_1	c6bbf34fdff071c1631dadcbfcebdcfc
K_2	5329a0a02908652c64781aee28f06a9e
K_3	10b1898d04e239bfa49e17ebe95a9197
K_4	5c91f36ade782c2690fab5e6f321eb51
A	00000000000000000000000000000000
B	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
	00000000000000000000000000000000
C	2c578f7927a949d3b511ae8fb69145c6

Variable	Value
X_1	5c6b03bab0976b641124560ed4ba2408
X_2	55867ce090a7597c4d3044ec6fbebac7
X_3	f991a437ce4192088d4b57d32ce81c46
X_4	10baacdf9eac69a95653ffd45fd9d878
X_5	6cc344dfbbe84e3489dbc61896bed197
X_6	49083f5790ba45227a6b6772ea9216a8
X_7	2077f91d78c8530c27ade9df22003d6b
X_8	38a02d2382aad97a0d05d652129e5854
X_9	c5682e43025bcf8d92a3351c08c3ee1f
X_{10}	6ba34cd6156cec249192e4e01c095d2e
X_{11}	406788525af56f7a0ef2b2951287529d
X_{12}	826dd62ed7d6bd6e4864460ad0564446
X_{13}	a4975e7da5e0d7496b33bbe0681f5733
X_{14}	bcc4009a8cda15f5755c81d95c15c3ba
X_{15}	e15a3b86f0e989bb45a2c05e1301d521
X_{16}	9b597090e3e4966607d3a35466c8161e
X_{17}	c6c383271c7d3a8dfa35f2bbf2364659
X_{18}	ab4e4468163fbd1a1dd2a0eee6a23cdb
X_{19}	fe44881cd2b9f743335c8537e780734b
X_{20}	94ff616e3fa2a49c2093af246f6cfbb0
X_{21}	2763b47eaebb44cc3b16075e9963f9d6
X_{22}	11112b53a1d3d249ec7a7a93ddb9a434
X_{23}	8503d52c54254fc5f1485309b614d29d
X_{24}	c35357d50432baa88721bfaee63705eb
X_{25}	c80b2f88d2f04a6eb8ddefca35efb09f
X_{26}	ed9afb52d66bef5fed7427e92b951e2c
X_{27}	47fdc11c4e6bd38eae07bc47bcd2c2c6
X_{28}	7117b6431e2781c6193d631151ab9363
X_{29}	0cefcbff686d397807f0c77bb6848ca6
X_{30}	5c915e9ea63ae07c06607e26347305df
X_{31}	758c0f79743a569ed49521dc3e50e490
X_{32}	dcdee25acacf602586f93555e2d61a0c
X_{33}	4889f1e2ab1c9c9586e45dbfcb799ffb
D	64de7e9b8cb5d54633f5f3307de8da3d

Variable	Value
Y_1	64de7e9b8cb5d54633f5f3307de8da3d
C_1	0d683b183a3e178ced28c7340175c7b6
Y_2	64de7e9b8cb5d54633f5f3307de8da3e
C_2	7611cf25eb0e9a2318b798c643d9815c
Y_3	64de7e9b8cb5d54633f5f3307de8da3f
C_3	723422504806dcf4d7892ca280d9c63b
Y_4	64de7e9b8cb5d54633f5f3307de8da40
C_4	b4230e84ff29ef575b525b82fabe023e
Y_5	64de7e9b8cb5d54633f5f3307de8da41
C_5	59cb9441d39a146aeee0c41bee6f51ea
Y_6	64de7e9b8cb5d54633f5f3307de8da42
C_6	dceba56600a6c503a611a201543758f1
Y_7	64de7e9b8cb5d54633f5f3307de8da43
C_7	2553571570f61b93a5e88f6044e8b49d
Y_8	64de7e9b8cb5d54633f5f3307de8da44
C_8	854aca2c455cf37a26bb56a81b736998
Y_9	64de7e9b8cb5d54633f5f3307de8da45
C_9	fca69195940476076bc4fdbedfa55cb8
Y_{10}	64de7e9b8cb5d54633f5f3307de8da46
C_{10}	46632a0fd4ab01b29a8e40519ffc476b
Y_{11}	64de7e9b8cb5d54633f5f3307de8da47
C_{11}	e992de544e435b33c8664e5e05281a0c
Y_{12}	64de7e9b8cb5d54633f5f3307de8da48
C_{12}	8ce8ffeb1f54d9cd13a4523caace820d
Y_{13}	64de7e9b8cb5d54633f5f3307de8da49
C_{13}	c07cc362b234989295f1384dedad9a37
Y_{14}	64de7e9b8cb5d54633f5f3307de8da4a
C_{14}	4ebe9cb3b9bb68f6bca463b13d2f128b
Y_{15}	64de7e9b8cb5d54633f5f3307de8da4b
C_{15}	3548cd1b1683e2aba097cc288dfe3bd1
Y_{16}	64de7e9b8cb5d54633f5f3307de8da4c
C_{16}	4b4b272ac5517bb17b86e52862ae8df4

Variable	Value
Y_{17}	64de7e9b8cb5d54633f5f3307de8da4d
C_{17}	d5e7683acee2bf39818d6a774935ae0e
Y_{18}	64de7e9b8cb5d54633f5f3307de8da4e
C_{18}	609cc97a21174b9f8543d2836b813ba6
Y_{19}	64de7e9b8cb5d54633f5f3307de8da4f
C_{19}	3d6c9778a076c71613a2da9cf857f73a
Y_{20}	64de7e9b8cb5d54633f5f3307de8da50
C_{20}	b02558a3f064861e7872c2889167cc78
Y_{21}	64de7e9b8cb5d54633f5f3307de8da51
C_{21}	d219f41fae7b511ea2b00aa219101678
Y_{22}	64de7e9b8cb5d54633f5f3307de8da52
C_{22}	6573424b1aa608e4188e053d708e5fc9
Y_{23}	64de7e9b8cb5d54633f5f3307de8da53
C_{23}	22462845bf932ef4995f25905fbfbace
Y_{24}	64de7e9b8cb5d54633f5f3307de8da54
C_{24}	79ff8398153e287f991946786ec33d38
Y_{25}	64de7e9b8cb5d54633f5f3307de8da55
C_{25}	bd5535a3f9be795ec9536907b76c0885
Y_{26}	64de7e9b8cb5d54633f5f3307de8da56
C_{26}	2554a62b304dcbc83ea60ac13d7571fd
Y_{27}	64de7e9b8cb5d54633f5f3307de8da57
C_{27}	15414a7d2bb6770b86434bc779f08285
Y_{28}	64de7e9b8cb5d54633f5f3307de8da58
C_{28}	f2a16a17d43a844b2e607ec26eb4e4e2
Y_{29}	64de7e9b8cb5d54633f5f3307de8da59
C_{29}	3b6547782ea97975315a0c835f2f8b99
Y_{30}	64de7e9b8cb5d54633f5f3307de8da5a
C_{30}	49ffa58d04d8c7e218609e817f737a8d
Y_{31}	64de7e9b8cb5d54633f5f3307de8da5b
C_{31}	14a24a064811e14ad5eef66714a71e6c

Variable	Value
E	0d683b183a3e178ced28c7340175c7b6 7611cf25eb0e9a2318b798c643d9815c 723422504806dcf4d7892ca280d9c63b b4230e84ff29ef575b525b82fabe023e 59cb9441d39a146aeee0c41bee6f51ea dceba56600a6c503a611a201543758f1 2553571570f61b93a5e88f6044e8b49d 854aca2c455cf37a26bb56a81b736998 fca69195940476076bc4fdbedfa55cb8 46632a0fd4ab01b29a8e40519ffc476b e992de544e435b33c8664e5e05281a0c 8ce8ffeb1f54d9cd13a4523caace820d c07cc362b234989295f1384dedad9a37 4ebe9cb3b9bb68f6bca463b13d2f128b 3548cd1b1683e2aba097cc288dfe3bd1 4b4b272ac5517bb17b86e52862ae8df4 d5e7683acee2bf39818d6a774935ae0e 609cc97a21174b9f8543d2836b813ba6 3d6c9778a076c71613a2da9cf857f73a b02558a3f064861e7872c2889167cc78 d219f41fae7b511ea2b00aa219101678 6573424b1aa608e4188e053d708e5fc9 22462845bf932ef4995f25905fbfbace 79ff8398153e287f991946786ec33d38 bd5535a3f9be795ec9536907b76c0885 2554a62b304dcbc83ea60ac13d7571fd 15414a7d2bb6770b86434bc779f08285 f2a16a17d43a844b2e607ec26eb4e4e2 3b6547782ea97975315a0c835f2f8b99 49ffa58d04d8c7e218609e817f737a8d 14a24a064811e14ad5eef66714a71e6c

Variable	Value
X'_1	bd709877bb44def9c17f4b9660e79034
X'_2	66c14200e418c1936df177c74423493c
X'_3	bd26575d4893af055501b42350f2b0f5
X'_4	8274c32c7a049c5f70155debd2b2ed7e
X'_5	a0c41b5a0a67e85b2392866ff644fbc3
X'_6	f866e6420f413c04bd45e2dd46c83773
X'_7	5f62d84a09dc7b03cfdf8c2960ae0414
X'_8	1ea481d9622affe26557e9ca47471a69
X'_9	d82dd0c6b5e34a3226bae62d3e6801b5
X'_{10}	5533a34ab8ab1af066f0e437fa276a45
X'_{11}	cfb7c995d32bfba53870bb803a3ff65a
X'_{12}	6c4531e73ae2ff2cc940c200651bd3a4
X'_{13}	f90a7a6e97ffe3d53568f27bb62ec8e4
X'_{14}	a2bb4aa5cf51c1e7f5feb183ecd0bd93
X'_{15}	d75694eac2af04fae8a18b4d45b982f7
X'_{16}	df8e6ac581e260667bc484882550c648
X'_{17}	a80c69989248e3d6e854753f0b3cb1e2
X'_{18}	c02911dfd11caed7a7e9bb80a87b0cbb
X'_{19}	6823ad9ff0dab54bbac6e8df27c09ea5
X'_{20}	c11f218b8a65e7a9c88485ea15b9b9dd
X'_{21}	bd5af3c996f5b885d10703475f0c11b8
X'_{22}	f2384e75b35b5515892b8a38ba876b49
X'_{23}	1aab27da4a9c150d3b0c170413fbfbbb
X'_{24}	439fcad79c17bf851a9383e0587f99e7
X'_{25}	8f7df83c4bee76ea74d408231f5c12d4
X'_{26}	44b577aee0f4c3297776c2f02792320a
X'_{27}	cfa2a342c1fe946bc027411f394caf57
X'_{28}	acf7cc8fe0e668e5f45bfd853ca2192e
X'_{29}	3777aa7734a8c140464d9c694455ce40
X'_{30}	d62049b32602c8dc4bb5a527b80e5d04
X'_{31}	35206767f2ba76b1fc25f177120492b9
X'_{32}	aea61a18ebc0895948da42447afbbd15
X'_{33}	46cba0b362da628d86d8ea2f16503476
F	2215de28ee6fb7cbb52d191f6bb8ee4b
G	55d00a317ddff00d731e03cdadaa6d81