The chain & sum primitive and its applications to MACs and stream ciphers

Mariusz H. Jakubowski[1] and Ramarathnam Venkatesan[2]

[1] Princeton University, mj@cs.princeton.edu
[2] Microsoft Research, Redmond, WA 98052, USA, venkie@microsoft.com

Abstract. We present a new scheme called *universal block chaining with sum* (or *chain & sum primitive (C&S)* for short), and show its application to the problem of combined encryption and authentication of data. The primitive is a weak CBC-type encryption along with a summing step, and can be used as a front end to stream ciphers to encrypt pages or blocks of data (e.g., in an encrypted file system or in a video stream). Under standard assumptions, the resulting encryption scheme provably acts as a random permutation on the blocks, and has message integrity features of standard CBC encryption. The primitive also yields a very fast message authentication code (MAC), which is a multivariate polynomial evaluation hash. The multivariate feature and the summing aspect are novel parts of the design. Our tests show that the chain & sum primitive adds approximately 20 percent overhead to the fastest stream ciphers.

1 Introduction

For combined encryption and authentication of data, one often uses stream ciphers because of their speed in comparison to block ciphers; one then appends a separately computed MAC value. However, in some applications, data must be accessed in pages or blocks, and stored encrypted, as in some encrypted file systems or video streams. For this purpose, it is customary to use CBC encryption on the blocks, and compute an integrity check for the entire stream separately from these individual encrypted blocks. Alternately, one may compute and store one MAC value separately per block, but this causes the size of the MAC data to expand in proportion to the number of blocks, and is thus undesirable. Certain applications, such as encrypted file systems and video, tax the CPU rather harshly, and using a block cipher can cause a noticeable performance hit. For backward compatibility and maximum efficiency, a different approach is required.

We would like to encrypt and access data in blocks or pages, and mimic CBC encryption on these block levels; furthermore, we would like to avoid increasing the lengths of the individual blocks, and keep one incremental MAC value for the entire stream. (We do not address the incremental MAC issue here.) In many applications, such as the ones cited above, that have implicit semantics, wrongly decrypted blocks will be immediately noticeable; in addition, the overall stream MAC must defend against the case in which the input data is completely random.

Here we present a new primitive called *universal block chaining with sum (C&S)*. In an application of C&S, a sequence of input data (which typically consists of hundreds of machine words) is first processed by a CBC-like primitive, in which the cipher is replaced by a pair of invertible universal hash functions that are applied alternately. The output words are then summed up and written in place of the next-to-last word. The last two words, called the *pre-MAC*, are encrypted with a block cipher; this encrypted value is implicitly a MAC value for the input data. We use the pre-MAC directly or indirectly to synchronize a stream cipher (e.g., [1]), or to generate a stream-cipher key, compute a pseudo-random one-time pad with the stream cipher, and encrypt the rest of the words with this pad.

We describe our main scheme and prove its security properties. From this scheme we derive variants that can be analyzed similarly. Then we show two different types of applications. First, using our scheme as a front end (as described above), we show that it mimics standard CBC encryption in terms of message integrity. Using standard assumptions about underlying ciphers, we show that our construction behaves as a random permutation on the entire input. This is advantageous, since stream ciphers are typically much faster than block ciphers. The design is intended for encryption of stored or streamed data that must be accessed and processed in pages or blocks. With a simple incremental scheme, one can easily compute MAC values for the entire stream. Heuristic considerations can be given to argue how C&S may remove weaknesses in existing ciphers [6] and strengthen the security of the stream cipher itself. We omit them for lack of space.

Our construction has the side effect of yielding ciphers that seem stronger than the stream cipher with which we begin. For example, if we combine the alleged RC4 stream cipher with our scheme, the second binary derivatives of the LSBs of the outputs are hard to infer, which is not true with the original alleged RC4 cipher [6]. Note that the stream cipher encrypts only the weakly encrypted output from the C&S stage, and the output of the alleged RC4 cipher is not released in the clear. It is an open problem to analyze this effect precisely.

A more detailed version of this paper will be available from the authors, or at `http://www.cs.princeton.edu/~mj/`.

Prior Work: One commonly uses a block cipher in CBC mode, which has been recently analyzed in [5] and shown to be secure in the ideal cipher model. However, CBC is very slow in software; in the applications mentioned earlier, CBC leads to an unacceptable performance hit. Another approach is to use collision-resistant hash functions (see ([3] for techniques and references). However, our schemes are faster than those methods; in addition, such methods do not offer a way to add local integrity at block level and to enhance the security of the stream cipher, as our scheme does.

Universal hash constructions are due to Carter and Wegman [16], and were used by Krawczyk [10], Rogaway [14], Shoup [15] and Halevi and Krawczyk [9].

Shoup's paper gives a comprehensive survey of all technical definitions needed here.

Our contributions: We introduce a new method for combining encryption and authentication, which also has the benefit of yielding a fast reversible MAC. We present and analyze our main construction for the chain & sum primitive and prove its security under standard assumptions. The analysis of the addition step's effect is quite novel, and the addition permits us to build $2l$-bit valued MACs from l-bit operations. This output-doubling problem is usually not easy to solve (see [15, 2]). Our primitive differs from related schemes in a striking way.

This in turn allows us to construct random permutations from nl bits to nl bits, given a $2l$-bit to $2l$-bit ideal cipher and a stream cipher with a $2l$-bit key. Our MAC operations work hand in hand with the stream ciphers and heuristically seem to have the effect of enhancing them. We would hope that this would be good front end to many stream ciphers which may have some minor defects. The MAC value of a block or page can be read off from the last $2l$-bits of the encrypted input.

1.1 Implementation and performance

Our construction yields a very efficient MAC, which uses only basic processor operations. Nonetheless, the speed of the MAC computations can depend significantly on their implementation on a particular processor; for example, when using simple instructions such as bit test, some modern processors with aggressive branch prediction incur a penalty of about 10 cycles when branches are mispredicted and the processor pipeline stalls. Hence, careful processor-specific hand optimization and testing of assembly-language code are needed to achieve maximum speed.

We report on the speeds of our main algorithm, described in section 2. Our implementations and tests were done on Pentium 200 MHz and 266 MHz systems. The computations operated on values in the fields $Z_{2^{31}-1}$ and $GF(2^{32})$, and computed 64-bit MACs. In $Z_{2^{31}-1}$, the resulting MAC has actually only 62 bits of security.

The overhead of adding the chain & sum computations to stream ciphers (RC4 and VRA) was between 10 and 45 percent, depending on which stream cipher was used and how key setup was done. On our Pentium 200 MHz and 266 MHz systems, the respective speeds of an assembly-language version of the chain & sum operations in the field $Z_{2^{31}-1}$ were approximately 360 Mbps and 470 Mbps when data were in cache, and 240 Mbps and 350 Mbps when data were in memory. The fastest 64-bit MAC of which we are aware, MMH [9], was reported to run at 500 Mbps and at 380 Mbps on in-cache and in-memory data, respectively, on a 200 MHz Pentium; however, unlike the chain & sum MAC, the MMH MAC does not seem to admit reversible implementations.

In the field $GF(2^{32})$, multiplication can be done using lookup tables, as in [15]. Finding irreducible polynomials in this field takes about $10\mu s$ on our Pentium 200 MHz machine, and is an order of magnitude faster than finding such polynomials in $GF(2^{64})$. Our schemes take advantage of this when generating a 64-bit MAC using only 32-bit operations.

In addition, the speed of our schemes may be increased significantly by exploiting the following observation: Given a randomly fixed $a \in GF(2^\ell)$, for all $x \in GF(2^\ell)$, $x \mapsto ax$ must be considerably easier to compute than the case when a is a variable. If x can be expressed uniquely as a linear combination of $1, a, a^2, \ldots, a^{\ell-1}$, then the map is computed by left shifting x and adding the bit string $c_{\ell-1}, \ldots, c_1, c_0$ where $a^\ell = \sum_i c_i a^i$. The analysis of using this observation in our constructions is surprisingly complicated and lengthy, and we do not present this implementation-related issue here.

1.2 Conventions and notation

All our algorithms operate on values belonging to some finite field F, typically $GF(2^{32})$ or Z_p for some large prime p. We use the period ('.') to denote concatenation of values such as the elements of $GF(2^{32})$, which are represented by bit strings. We use $X = x_0.x_1...x_n$ to denote a plaintext message consisting of $n+1$ l-bit blocks x_0, x_1, ..., x_n, where $n = 2m+1$. We assume that X contains an even number of blocks.

Let $h(x)$ denote an ideal cipher (random permutation), where x represents a two-block ($2l$-bit) input, and let $G(s)$ represent an ideal stream cipher.

2 The chain & sum primitive and its use in encryption

We now describe a typical encryption scheme that uses the chain & sum primitive. In later sections, we describe variants of this basic algorithm.

Let the encryption key K be composed of six l-bit strings a, b, c, d, e, and e'. Define $f(x) = ax + b$ and $g(x) = cx + d$. The first two steps below show an application of chain & sum. We define $CS_K(C) = y_0, \ldots, y_n$.

1. Let $C = c_0.c_1...c_n$, where $c_0 = f(ex_0)$, $c_i = f(c_{i-1} + ex_i)$ for even $i > 0$, and $c_i = g(c_{i-1} + e'x_i)$ for odd $i \geq 1$.
2. Let $y_k = c_k$ for $k \neq n-1$, $y_{n-1} = \sum_{k=0}^{n} c_k$, and $s = y_{n-1}.y_n$.
3. Replace $y_{n-1}.y_n$ with the encryption of s, namely $h(s)$.
4. Let $Z = G(s) \oplus y_0.y_1...y_{n-2}$ denote the result of encrypting Y with the pseudorandom sequence $G(s)$. Here $\oplus$ denotes bitwise exclusive OR of two quantities of equal length in binary.

We refer to the string s as the *pre-MAC*, and to the string $y_{n-1}.y_n$ after the step (3) above as the MAC value of X.

3 Security of the basic scheme

3.1 Preliminaries

Let $H_w(X)$ be an indexed (by w) family of functions mapping X to $2l$ bits. A random member of the family is chosen by picking a random w, which normally here would correspond to the secret keys. We say that such a random member f is *pairwise independent* if for any pair of distinct x_1 and x_2 we have that $\mathbf{Pr}[f(x) = \Omega] = 2^{-2\ell}$, and $\mathbf{Pr}[f(x_2) = \Omega' | f(x_1) = \Omega] = \mathbf{Pr}[f(x_2) = \Omega']$, for any Ω, Ω' in the range. We say that the family has collision probability ϵ if for a random w and any pair X, X', the probability that $H_w(X) = H_w(X')$ is at most ϵ. We will model stream ciphers or secure pseudo-random number generators by a function $G(s) = R$ that maps $2l$ bit inputs s to $(n-2)l$-bit strings R randomly. We will assume that G is ideal and behaves like a random function.

We use the ideal block cipher model when analyzing DES or other related ciphers. In this model, $DES_{K'}(m) = e$ will be treated as a family of independent random permutations indexed by K', each mapping $2l$ bits to $2l$ bits.

Let $E_K(X) = Y$ be a family of permutations mapping nl-bits to nl-bits. We say $E_K(.)$ has *local integrity* if the mapping is a random permutation from $|X| = nl$ bits to nl-bits. In such a case, if one changes any bit of a given input X to obtain X', this will result in completely random output; that is, $E_K(X)$ and $E_K(X')$ will be statistically uncorrelated or independent. A desirable aspect of traditional CBC encryption is that if one bit of the encrypted message is changed, decryption will result in one garbled block and a bit error in another block. However, local integrity does not imply integrity in the usual sense; it fails if the input data is random. We consider the problem of achieving local integrity and mimicking the properties of CBC encryption in terms of the effects that ciphertext modification has on the decrypted inputs.

Now we define our encryption scheme $E_K(X_i)$. To encrypt X_i, we first compute $CS_K(X_i) = Y_i$, and then encrypt the last $2l$ bits of Y_i, denoted by $\bar{h}_i$, with an ideal cipher to obtain z_i. Then we compute $G(\bar{h}_i) = R'_i$ of length $(n-2)l$ bits, and bitwise-XOR this with the first $(n-2)l$ bits of $CS_K(X_i)$ to obtain R_i. We output R_i, z_i. Let $\mathcal{A}$ be an adversary who has unbounded resources, obtains an nl-bit input, and outputs either 0 or 1. Let μ, λ be two probability distributions on nl bit strings. We write $X \leftarrow \mu(.)$ if X_i is chosen according to the distribution $\mu(.)$. Then we say that $\mathcal{A}$ distinguishes the distributions λ and μ with advantage ε if $\mathbf{Pr}[(X_i) = 1 | X_i \leftarrow \mu(.)]$ and $\mathbf{Pr}[(X_i) = 1 | X_i \leftarrow \lambda(.)]$ differ at most by ε.

Theorem 1. *The encryption function $E_K(X_i) = Y$ cannot be distinguished from truly random permutations mapping nl bits to nl bits with probability better than 2ε, using adaptive attacks with $q \leq 2^\ell$ chosen plaintext-ciphertext pairs. Here ε is the probability of a collision among the MAC values $\bar{h}_i, i \leq q$.*

Proof (outline): First, we note that if all the MAC values $\bar{h}_1, \bar{h}_2, \ldots, \bar{h}_q$ corresponding to the q queries are distinct, then the distribution of the outputs of $E_K(.)$ is identical to that of a perfect random permutation. Secondly, we bound the probability of the collision as required. The bulk of the paper is devoted to this latter task.

If the $\bar{h}_i$'s are distinct, then the z_i's are independent outputs of a random permutation and R'_i are outputs of a random function. The random function and the random permutation are independent. Note that the outputs of the random permutation are all distinct, but we perform fewer than 2^l queries, and thus a truly random function would output distinct strings as well with probability $1-\varepsilon$. From this the indistinguishability follows.

Overview: Lemmas 1 through 5 are towards bounding the collision probability of the MACs in the above theorem. The subsection 3.2 presents an algorithm which has a quadratically smaller fraction of bad keys. Finally, in section 4, we give another algorithm variant and analyze its collision probability.

Let $C = c_0c_1...c_n$ represent the result of one chaining step performed on X, as in step 1 of the encryption algorithm; that is, let $c_0 = f(ex_0)$, $c_i = f(c_{i-1} + ex_i)$ for even $i > 0$, and $c_i = g(c_{i-1} + e'x_i)$ for odd $i > 0$. Let $\alpha = ac$ and $\beta = ae$. We use the convention $\sum_{i=a}^{b} F(i) = 0$ if $a > b$.

Let $Y = y_0y_1...y_n$ denote the result of step 2 of the encryption algorithm; that is, let $y_i = c_i$ for $i \neq n-1$, and $y_{p-1} = \sum_{k=0}^{n} c_k$.

For $0 \leq r \leq m$, we obtain the following:

Lemma 1.

$$c_{2r} = b + \beta x_{2r} + a\sum_{k=0}^{r-1}\alpha^k(d+bc) + \sum_{k=1}^{r}\alpha^k(\beta x_{2r-2k} + e'x_{2r-2k+1}) \qquad (1)$$

$$c_{2r+1} = \sum_{k=0}^{r}\alpha^k(d+bc) + c\sum_{k=0}^{r}\alpha^k(\beta x_{2r-2k} + e'x_{2r-2k+1}) \qquad (2)$$

Proof. We use induction on r. For $r = 0$, eqs. 1 and 2 give $c_0 = b + \beta x_0$ and $c_1 = d + c(b + \beta x_0 + e'x_1)$, so the base case holds. Assume eqs. 1 and 2 are true for $r \leq p$. For $r = p+1$, we have

$$\begin{aligned}
c_{2(p+1)} &= a(c_{2p+1} + ex_{2(p+1)}) + b \\
&= b + \beta x_{2(p+1)} + a\sum_{k=0}^{p}\alpha^k(d+bc) + \sum_{k=0}^{p}\alpha^{k+1}(\beta x_{2p-2k} + e'x_{2p-2k+1}) \\
&= b + \beta x_{2(p+1)} + a\sum_{k=0}^{p}\alpha^k(d+bc) + \sum_{k=1}^{p+1}\alpha^k(\beta x_{2(p+1)-2k} + e'x_{2(p+1)-2k+1}) \\
&= c_{2(p+1)}
\end{aligned}$$

and

$$\begin{aligned}
c_{2(p+1)+1} &= c(c_{2(p+1)} + e'x_{2(p+1)+1}) + d \\
&= c\beta x_{2(p+1)} + ce'x_{2(p+1)+1} + d + bc + \sum_{k=0}^{p}\alpha^{k+1}(d+bc) + \\
&\quad + c\sum_{k=1}^{p+1}\alpha^k(\beta x_{2(p+1)-2k} + e'x_{2(p+1)-2k+1}) \\
&= \sum_{k=0}^{p+1}\alpha^k(d+bc) + c\sum_{k=0}^{p+1}\alpha^k(\beta x_{2(p+1)-2k} + e'x_{2(p+1)-2k+1}) \\
&= c_{2(p+1)+1}.
\end{aligned}$$

Thus, eqs. 1 and 2 are verified. □

Pairwise independence of last blocks Let $X' = x'_0 x'_1 ... x'_n$ denote a plaintext different from X, and let $Y' = y'_0 y'_1 ... y'_n$ represent the result of step 2 of the encryption algorithm performed on X'. We show that the following holds:

Lemma 2. *The mapping $X \mapsto y_n$ defined above is an almost 2-universal hash function. That is, for arbitrary Ψ and Ω, $\mathbf{Pr}[y_n = \Psi] = 2^{-l}$, and there exists a $1 - 4m/2^l$ fraction of coin flips (for the choice of the key K) for which $\mathbf{Pr}[y_n - y'_n = \Omega | y_n = \Psi] = \mathbf{Pr}[y_n - y'_n = \Omega]$.*

Proof. We will show that given two plaintexts, X and X', the random variables y_n and y'_n are pairwise independent when α is set appropriately. That is, given any Ψ and Ω from F, we shall find values of a, b, c, d, e, and e' such that both $y_n = \Psi$ and $y_n - y'_n = \Omega$.

Let $\delta_i = x_i - x'_i$. The equation $y_n - y'_n = \Omega$ can be written as

$$e \sum_{k=0}^{m} \alpha^{k+1} \delta_{2m-2k} + ce' \sum_{k=0}^{m} \alpha^k \delta_{2m-2k+1} = \Omega \tag{3}$$

where the values of Ω and $\delta_i, 0 \leq i \leq m$, are fixed. We may represent eq. 3 as

$$eP_1(\alpha) + ce'P_2(\alpha) = \Omega \tag{4}$$

where the polynomials P_1 and P_2 correspond to the summations in eq. 3. For technical reasons mentioned later, we assume that our choice of α satisfies $P_3(\alpha) = \alpha^{m+1} - 1 \neq 0$. In addition, α will satisfy some conditions relative to P_1 and P_2 mentioned next. We will call such α's *good.*

If X and X' are distinct then, the polynomials P_1 and P_2 cannot be both trivial (i.e. vanish at all points). Thus, for some i, P_i is not trivial. Note that there are at most m values of α that are roots of P_1 or P_2; thus, we can find a value α such that at least one of $P_i(\alpha)$ is non-zero.

Assuming we have chosen a good α, we substitute its value in eq. 3, obtaining the two-variable linear equation

$$eC_1 + ce'C_2 = \Omega \tag{5}$$

where C_1 and C_2 are constants equal to P_1 and P_2, respectively, evaluated at α.

We may now compute the values of a, c, e and e' as follows:

1. If $C_1 = 0$ and $C_2 \neq 0$, for every choice of e, one can set $ce' = \Omega C_2^{-1}$, and $a = \alpha c^{-1}$.
2. If $C_1 \neq 0$ and $C_2 = 0$, then set $e = \Omega C_1^{-1}$. In this case any choice of $a, ce' \in F$ such that α is good will suffice.
3. If $C_1 \neq 0$ and $C_2 \neq 0$, then set $ce' = (\Omega - eC_1)C_2^{-1}$ and a is such that α is good.

Thus, we can assume we have picked values of a, c, e, and e' to set $y_n - y'_n = \Omega$, given that δ_i are fixed for $0 \leq i \leq n$. Since $P_3(\alpha) \neq 0$, we can pick b and d to satisfy $y_n = \Psi$ as follows. By eq. 2, we have

$$y_n = \sum_{k=0}^{m} \alpha^k (d + bc) + C \tag{6}$$

$$= (d + bc)\frac{\alpha^{m+1} - 1}{\alpha - 1} + C \tag{7}$$

where C is a constant. Thus, if $\alpha^{m+1} - 1 \neq 0$, we can choose values of b and d to set y_n to any value.

It is easily seen that the choices we made for the values of the parameters will occur with uniform probability, thus yielding the lemma. The number of the offending cases of α which violate the conditions relative to $P_i, i \leq 2$ is at most $4m$, which bounds the error probability as required. □

Note that we may modify the basic scheme to use $f(x) = ax$ and $g(x) = cx$ as the functions to be chained, and to add b and d to the values of the next-to-last (y_{n-1}) and last (y_n) blocks, respectively. This allows us to use b and d more easily to prove the pairwise-independence properties in the above proof, and speeds up computations slightly in implementations; however, these changes appear to decrease the heuristic effectiveness of the primitive in enhancing the stream cipher (although we have not fully analyzed the effect of the primitive on the stream cipher). For example, when a block of data consisting of all 0's is processed by the primitive in this case, the result is a block consisting mostly of 0's; this is not true with our original scheme.

Collisions in the last and next-to-last blocks

The need for the sum Using two-function block chaining is natural, and one may expect that the last two l-bit blocks must result in a good pre-MAC. If this were the case, one would expect that on two distinct inputs, X, X' the events, "there is a collision in the last block of the output" and "there is a collision in the next-to-last blocks" be uncorrelated. But this is not true: If two inputs X and X' do not differ in the last block, then the collisions of y_n and y'_n are strongly correlated with those of c_{n-1} and c'_{n-1}:

$$\mathbf{Pr}[c_n = c'_n | c_{n-1} = c'_{n-1}, \delta_n = 0] = \mathbf{Pr}[c_{n-1} = c'_{n-1} | c_n = c'_n, \delta_n = 0] = 1 \tag{8}$$

Decoupling the correlations Given that $c_n = c'_n$, and thus $\delta_n = 0$, our addition construction will hope to produce y_n, y_{n-1} so that these variables have their collision probabilities decoupled; since there must be some $j < n$ such that j-th blocks of inputs X and X' are distinct, there must be some $c_w, c'_w, w < n$ which do not collide, and the addition may break up the collisions in the last but one block. This is what we prove next.

Let $\Delta = y_{2m}$ and $\Delta' = y'_{2m}$. We show the following:

Lemma 3. *When α is good,* $\mathbf{Pr}(\Delta = \Delta' | y_n = y'_n) = \mathbf{Pr}[\Delta = \Delta'] = 1/|F|$.

Proof. Given two plaintexts, X and X', we show how to choose values of a, b, c, d, and e such that $\Delta - \Delta' = \Omega$, for any $\Omega \in F$, given that $c_n - c'_n = 0$. Assume $e' = 1$. Note that $y_n = y'_n$ if and only if $c_{n-1} - c'_{n-1} = c\delta_n$. Hence $\mathbf{Pr}[\Delta - \Delta' = \Omega | y_n = y'_n] = \mathbf{Pr}[\sum_{j=0}^{n-2} c_j + c_{n-1} - \sum_{j=0}^{n-2} c_j - c'_{n-1} = \Omega | c_{n-1} - c'_{n-1} = c\delta_n]$.

After one C&S step is performed on the plaintext X, the next-to-last block of the result is given by

$$\Delta = \sum_{i=0}^{m} c_{2i} + c_{2i+1}. \tag{9}$$

A similar formula holds for Δ'.

The equation $\Delta - \Delta' = \Omega$ can be written as

$$\sum_{i=0}^{m} c_{2i} - c'_{2i} + c'_{2i+1} - c'_{2i+1} = \Omega \tag{10}$$

which is equivalent to

$$\sum_{r=0}^{m} \left(\beta\delta_{2r} + \sum_{k=1}^{r} \alpha^k (\beta\delta_{2r-2k} + \delta_{2r-2k+1}) + c \sum_{k=0}^{r} \alpha^k (\beta\delta_{2r-2k} + \delta_{2r-2k+1}) \right) = \Omega \tag{11}$$

The above can be rewritten as

$$\sum_{r=0}^{m} \left(\beta\delta_{2r} + \beta \sum_{k=1}^{r} \alpha^k \delta_{2r-2k} + \sum_{k=1}^{r} \alpha^k \delta_{2r-2k+1} + c\beta \sum_{k=0}^{r} \alpha^k \delta_{2r-2k} + c \sum_{k=0}^{r} \alpha^k \delta_{2r-2k+1} \right) = \Omega \tag{12}$$

which can be expressed as

$$\beta(C_0 + C_1 + cC_3) + C_2 + cC_4 = \Omega \tag{13}$$

where

$$C_0 = \sum_{r=0}^{m} \delta_{2r}, C_1 = \sum_{r=0}^{m} \sum_{k=1}^{r} \alpha^k \delta_{2r-2k}, C_2 = \sum_{r=0}^{m} \sum_{k=1}^{r} \alpha^k \delta_{2r-2k+1}$$
$$C_3 = \sum_{r=0}^{m} \sum_{k=0}^{r} \alpha^k \delta_{2r-2k}, C_4 = \sum_{r=0}^{m} \sum_{k=0}^{r} \alpha^k \delta_{2r-2k+1}$$

Letting $D_1 = C_0 + C_1 + cC_3$ and $D_2 = C_2 + cC_4$, we have

$$\beta D_1 + D_2 = \Omega. \tag{14}$$

We consider D_1 and D_2 as polynomials in α. After some manipulation, we write D_1 and D_2 as

$$D_1 = \sum_{i=0}^{m} \left(1 + c + (1+c) \sum_{k=1}^{m-i} \alpha^k \right) \delta_{2i} \tag{15}$$

$$D_2 = \sum_{i=0}^{m} \left(c + (1+c) \sum_{k=1}^{m-i} \alpha^k \right) \delta_{2i+1}. \qquad (16)$$

We note that for each j, the coefficient of each δ_{2j} and δ_{2j+1} in eqs. 15 and 16 is a polynomial of different degree (namely $m-j$). Then we use the following:

Lemma 4. *Let $f_1, f_2, \ldots, f_r \in F[x]$ be polynomials of distinct degrees. Then f_i are linearly independent over F. That is, any nontrivial linear combination $\sum \gamma_i f_i$, where $\gamma_i \in F$, is a nontrivial polynomial of degree $\geq min_j(deg(f_j))$.*

Proof: Consider the matrix where each row represents the coefficients of the polynomial. Since each degree is distinct in every column, every entry below a non-zero entry is zero. Consequently, we can delete all-zero columns (if any) and re-write the matrix as an upper triangular matrix. Such a matrix always has full rank. □.

Thus the polynomial for D_1 (i.e., $f \in F[x]$ such that $f(\alpha) = D_1$) cannot be trivial unless the inputs coincide in all odd numbered blocks. Similarly, the polynomial for D_2 cannot be trivial (i.e. identically zero for all values of α) unless the inputs agree on all even numbered blocks. Since X, X' are distinct, it follows at least one of these polynomials are nontrivial.

Thus, we can choose values for α and β such that substituting them in eq. 14 allows us to solve for nonzero c. We then compute $a = \alpha c^{-1}$, followed by $e = \beta a^{-1}$, and choose values for b and d arbitrarily. □

Collision probability of last block tuples We use the two lemmas above to show the following:

Lemma 5. *When α is good,* $\mathbf{Pr}(y_{2m} = y'_{2m}, y_{2m+1} = y'_{2m+1}) = 2^{-2l}$.

Proof.

$$\begin{aligned}\mathbf{Pr}(y_{2m} = y'_{2m}, y_{2m+1} = y'_{2m+1}) &= \mathbf{Pr}(y_{2m} = y'_{2m} | y_{2m+1} = y'_{2m+1}) \mathbf{Pr}(y_{2m+1} = y'_{2m+1}) \\ &= 2^{-2l}. \square\end{aligned}$$

Here we use the standard facts about the collision probabilities of pairwise independent hash functions and their approximations. (See [15] for details.)

3.2 Improving security of the algorithm

We modify step 1 of the encryption algorithm as follows:

1. Let $C = c_0 c_1 \ldots c_n$, where $c_0 = f(ex_0)$, $c_i = f(c_{i-1} + e^{i/2+1} x_i)$ for even $i > 0$, and $c_i = g(c_{i-1} + x_i)$ for odd $i > 1$.

Eqs. 1 and 2 become

$$c_{2r} = b + ae^{r+1} x_{2r} + a \sum_{k=0}^{r-1} \alpha^k (d + bc) + \sum_{k=1}^{r} \alpha^k (ae^{r-k+1} x_{2r-2k} + x_{2r-2k+1}) \quad (17)$$

$$c_{2r+1} = \sum_{k=0}^{r} \alpha^k (d + bc) + c \sum_{k=0}^{r} \alpha^k (ae^{r-k+1} x_{2r-2k} + x_{2r-2k+1}). \tag{18}$$

The expression $y_n - y'_n$ can now be written as

$$\sum_{k=0}^{m} \alpha^{k+1} e^{m-k+1} \delta_{2m-2k} + c \sum_{k=0}^{m} \alpha^k \delta_{2m-2k+1} \tag{19}$$

$$= e^{m+2} \sum_{k=0}^{m} \gamma^{k+1} \delta_{2m-2k} + c \sum_{k=0}^{m} \alpha^k \delta_{2m-2k+1} \tag{20}$$

where $\gamma = \alpha / e$.

The following stronger version of lemma 2 can be proved:

Lemma 6. *The mapping $X \mapsto y_n$ defined above is an almost 2-universal hash function. That is, for arbitrary Ψ and Ω, $\mathbf{Pr}[y_n = \Psi] = 2^{-l}$, and there exists a $1 - 4m^2/2^{2l}$ fraction of coin flips (for the choice of the key K) for which $\mathbf{Pr}[y_n - y'_n = \Omega | y_n = \Psi] = \mathbf{Pr}[y_n - y'_n = \Omega]$.*

Proof. As in the proof of lemma 2, given any Ψ and Ω from F, we shall find values of a, b, c, d, and e such that both $y_n = \Psi$ and $y_n - y'_n = \Omega$. Eq. 3 becomes

$$e^{m+2} \sum_{k=0}^{m} \gamma^{k+1} \delta_{2m-2k} + c \sum_{k=0}^{m} \alpha^k \delta_{2m-2k+1} = \Omega. \tag{21}$$

This can be written as

$$e^{m+2} P_1(\gamma) + c P_2(\alpha) = \Omega \tag{22}$$

where the polynomials P_1 and P_2 correspond to the summations in eq. 21. Note that their variables γ and α are independent variables and hence the probability that both are satisfied for random choice of the variables is $\leq m^2/2^{2l}$. If X and X' are distinct, then at least one of the polynomials P_1 and P_2 is nontrivial. Thus, we can choose a value of γ or α such that the left side of eq. 21 is nonzero; we then solve for e or c, compute a, and find b and d exactly as in the proof of lemma 2. In order to solve eq. 21 for e, it must be true that $gcd(|F|-1, m+2) = 1$, which can be achieved by choosing an appropriate field F and value of m.□

It can be shown that the following still holds, provided an appropriate field F is chosen.

Lemma 7. *When α is good,* $\mathbf{Pr}(\Delta = \Delta' | y_n = y'_n) = \mathbf{Pr}[\Delta = \Delta'] = 1/|F|$.

Proof. As in the proof of lemma 3, we show how to choose values of a, b, c, d, and e such that $\Delta - \Delta' = \Omega$, for any $\Omega \in F$, given that $c_n - c'_n = 0$.

The equation $\Delta - \Delta' = \Omega$ can be written as

$$\beta D_1 + D_2 = \Omega \tag{23}$$

where

$$D_1 = \sum_{r=0}^{m} \left(e^r (1 + c + (1+c) \sum_{k=1}^{m-r} \gamma^k) \right) \delta_{2r} \tag{24}$$

$$D_2 = \sum_{i=0}^{m} \left(c + (1+c) \sum_{k=1}^{m-i} \alpha^k \right) \delta_{2i+1}. \tag{25}$$

As in the case of lemma 3, the coefficients of δ_i's in the expression for D_1 (and D_2 similarly) are polynomials of distinct degrees. Similar analysis yields the lemma. □

4 A variant of the main algorithm

We now consider the following encryption algorithm. Its proof of security holds whether or not arithmetic is done using the finite-field scheme presented earlier. This variant differs from the main algorithm in that every l-bit block is processed twice.

Let the encryption key K be composed of four integers, a, b, c, and d. Define $f(x) = ax + b$ and $g(x) = cx + d$. The algorithm is as follows:

1. Let $C = c_0.c_1...c_n$, where $c_0 = f(x_0)$, and $c_i = f(c_{i-1} + x_i)$.
2. Let $Y = y_0y_1...y_n$, where $y_0 = g(x_0)$, $y_i = g(y_{i-1} + x_i)$, and $y_n = c_n$.
3. Replace y_{n-1} with $\sum_{k=0}^{n} y_k$.
4. Let $s = y_{n-1}.y_n$.
5. Replace $y_{n-1}.y_n$ with $h(s)$.
6. Let $Z = G(s) \oplus y_0.y_1...y_{n-2}$ denote the result of encrypting Y with the pseudorandom sequence $G(s)$. Here $\oplus$ denotes bitwise exclusive OR of two quantities of equal length in binary.

We refer to the string s as the *MAC* of the message. The next two lemmas estimate the probability of collisions. Owing to space considerations, we omit the proofs.

Lemma 8. *The mapping $X \mapsto y_n$ defined above is an almost 2-universal hash function. That is, for arbitrary Ψ and Ω, $\mathbf{Pr}[y_n = \Psi] = 2^{-l}$, and there exists a $1 - 4m^2/2^{2l}$ fraction of coin flips (for the choice of the key K) for which $\mathbf{Pr}[y_n - y'_n = \Omega | y_n = \Psi] = \mathbf{Pr}[y_n - y'_n = \Omega]$.*

Let $\Delta = y_{n-1}$ and $\Delta' = y'_{n-1}$.

Lemma 9. *When α is good, $\mathbf{Pr}(\Delta = \Delta' | y_n = y'_n) = \mathbf{Pr}[\Delta = \Delta'] = 1/|F|$.*

5 Conclusion

We have presented a new cryptographic primitive, chain & sum, with which we can efficiently add integrity to stream ciphers, as well as compute very fast MACs. The primitive can be used to design new encryption and authentication algorithms with provable security properties, and to create algorithms custom-tailored to run fast on machines with specific characteristics. When the operations of chain & sum are implemented in finite fields, the resulting scheme yields one of the fastest known MACs.

Acknowledgements: We thank Peter Montgomery for suggestions on implementing finite-field arithmetic, and Gideon Yuval for discussions related to hardware performance. We thank the anonymous Eurocypt98 referee for comments.

References

1. W. Aiello, S. Rajagopalan, R. Venkatesan, "Design and analysis of provably good random number generators," ACM SODA-95, pp. 1-9.
2. W. Aiello, R. Venkatesan, "Foiling birthday attacks in output doubling transformations," Advances in Cryptology –Eurocrypt 96.
3. M. Bellare, R. Canetti, H. Krawczyk, "Keying hash functions for message authentication," Advances in Cryptology–Crypto '96.
4. M. Bellare, R. Guérin, P. Rogaway, "XOR MACs: New methods for message authentication using finite pseudorandom functions," Advances in Cryptology–Crypto '95, pp. 15-28.
5. M. Bellare, J. Kilian, P. Rogaway, "On the security of cipher block chaining," Advances in Cryptology–Crypto '94, pp. 341-358.
6. J. Golic, "Linear Statistical Weaknesses in Alleged RC4 Keystream Generator," Advances in Cryptology–Eurocrypt '97, pp. 226-238.
7. A. Bosselaers, R. Govaerts, J. Vandewalle, "Fast hashing on the Pentium," Advances in Cryptology–Crypto '96.
8. G. Brassard, "On computationally secure authentication tags requiring short secret shared keys," Advances in Cryptology–Crypto '82, pp. 79-82.
9. S. Halevi, H. Krawczyk, "MMH: Software message authentication in the Gbit/second rates," Fast Software Encryption Workshop, 1996.
10. H. Krawczyk, "LFSR-based hashing and authentication," Advances in Cryptology–Crypto '94, pp. 129-139.
11. H. Krawczyk, "New hash functions for message authentication," Advances in Cryptology–Crypto '95, pp. 301-310.
12. J. Kilian, P. Rogaway, "How to protect DES against exhaustive search," Advances in Cryptology–Crypto 96.
13. B. Preneel, P. van Oorschot, "MDx-MAC and building fast MACs from hash functions," Advances in Cryptology–Crypto '95, pp. 1-14.
14. P. Rogaway, "Bucket hashing and its application to fast message authentication," Advances in Cryptology–Crypto '95, pp. 29-42.
15. V. Shoup, "On fast and provably secure message authentication based on universal hashing," Advances in Cryptology– Eurocrypt96. Later versions available from the author.
16. M. Wegman, L. Carter, "New hash functions and their use in authentication and set equality," Journal of Computer and System Sciences, 22:265-279, 1981.