The Mundja Streaming MAC

Philip Hawkes[1], Michael Paddon[1], and Gregory G. Rose[1]

Qualcomm Australia, Level 3, 230 Victoria Rd, Gladesville, NSW 2111, Australia
{phawkes,mwp,ggr}@qualcomm.com

Abstract. Mundja is a MAC generation algorithm that has been designed for use together with a stream cipher. Mundja accumulates the message onto two independent registers: the first is a Cyclic Redundancy Checksum (CRC) that uses linear feedback; the second is a strengthened version of the SHA-256 register [5] that uses nonlinear feedback. Mundja is fast (asymptotically about 4 times the speed of HMAC-SHA-256), and can generate MACs of any desired length. Mundja is designed to be secure at the equivalent level of 128-bit keys. When used in cooperation with a correspondingly secure stream cipher, it is hoped to remain secure even at the equivalent level of 256-bit keys. Appendices give details of the use of Mundja with the SOBER-128 [10], Turing [9] and RC4 stream ciphers.

1 Introduction

A Message Authentication Code (MAC, also occasionally known as a Message Integrity Check) is a tag, attached to a message by the sender, in order that the receiver with the same shared key can verify the identity of the sender and the integrity of the message. Mundja[1] is a MAC generation algorithm that has been designed for use with a stream cipher.

The intuition behind the creation of Mundja is that a MAC should be able to be more efficient than a corresponding hash function, because it is possible to utilize key-based secret state information to aid in resisting attacks, whereas a hash function typically has entirely known state. Constructions like HMAC [16] have to do extra work to overcome this "openness". Further, parallels with stream ciphers and recent results against hash functions [2, 15, 20] that have cast some doubts on the Davies-Meyer [17] iterated block hash function structure, lead us to a construction that does not have implicit blocks.

Mundja accumulates the message into two independent registers: the first is a 256-bit Cyclic Redundancy Checksum (CRC) that uses linear feedback; the second is a strengthened version of the SHA-256 register [5] that uses nonlinear feedback. Unlike the SHA-256 algorithm, there is no message expansion or separation into blocks. The security corresponding to the message expansion lies in finally combining these orthogonal registers. Information from the stream cipher state is used to initialize both of these registers, and instead of constants, further unknown input from the stream cipher state is taken as input to the SHA-256 round function. During analysis of the Mundja round function, we noticed [11] that the SHA-256 round function is not strong enough for the purposes of Mundja, in the sense of it being difficult to input different texts that cancelled out to create a collision in the SHA register, so Mundja alters the round function slightly by the introduction of a nonlinear S-Box to speed diffusion.

[1] From the word "mung": *muhng* (MIT, 1960) Mash Until No Good. Sometime after that the derivation from the recursive acronym "Mung Until No Good" became standard. To make changes to a file, especially large-scale and irrevocable changes.

2 Preliminaries

We introduce notation and reprise the design of SHA-256, so we can describe Mundja in terms of the parts that are the same, and those that are different. We also discuss the theory behind the Mundja CRC register.

2.1 Notation

Mundja and SHA-256 are based on 32-bit words. Within each word, the most significant bit (MSB) is the leftmost bit while the least significant bit (LSB) is the rightmost bit. Where words must be formed from octet-oriented data, Mundja uses Least Significant Byte first (little endian, c.f. Intel 80386), whereas SHA-256 uses Most Significant Byte first (big endian, c.f. SPARC).

The i-th bit of a word a is denoted $a[i]$. SHA-256 uses two bit-wise operators: "$\wedge$" represents the bitwise AND operation with $(a \wedge b)[i] = a[i] \wedge b[i]$, $0 \leq i \leq 31$; and "$\oplus$" represents the bitwise exclusive-OR operation with $(a \oplus b)[i] = a[i] \oplus b[i]$, $0 \leq i \leq 31$. The bit-wise complement of x (equal to $2^{32} - 1 - x$) is denoted x'. The function $ROTR^r(X)$ produces a word of the same size as X, but with the bits rotated cyclically to the right by r positions. That is, if $Y = ROTR^r(X)$, then $Y[i] = X[i + r(\bmod 32)]$, $0 \leq i \leq 31$. For any word X, let $\hat{X}$ denote the value of X with the MSB set to zero. Finally, for our analysis we will denote the Hamming weight of x (that is, the number of ones in the binary representation of x) by $|x|$.

2.2 Description of SHA-256

Padding: The message is padded and has its length in bits appended to make a multiple of 512 bits.

Parsing: The padded message is parsed into 512-bit blocks, $M^{(1)}, \ldots, M^{(N)}$. Each 512-bit input block is expressed as sixteen 32-bit words $M_0^{(i)}, \ldots, M_{15}^{(i)}$.

Message Expansion: The message expansion is applied to each message block individually. This is similar in principal to the key scheduling for a modern block cipher. The message expansion first assigns the message words $M_0^{(i)}, \ldots, M_{15}^{(i)}$ to the values of the input words $W_0, \ldots, W_{15}$. The remainder of the input words $W_{16}, \ldots, W_{63}$ are determined using an (almost) linear recurrence formula. We omit details, as this is not used by Mundja.

Register Update: The register has 8 words of state A, B, C, D, E, F, G, H. For the first block of the message, these words are initialized to pre-determined constants. For the remaining blocks of the message, the words are initialized to the intermediate hash value that results from the preceding message block. Following initialization, 64 rounds of the round function are applied to the expanded input sequence $\{W_t\}$. The t-th round of the round function modifies the register using input word W_t and a pre-determined constant K_t as input. The round function uses addition modulo 2^{32} and four functions: $CH, MJ, \Sigma_0, \Sigma_1$ with 32-bit inputs and 32-bit outputs, that are non-linear with respect to modular addition. These functions are defined as:

$$\begin{aligned}
CH(X, Y, Z) &= (X \wedge Y) \oplus (X' \wedge Z); \\
MJ(X, Y, Z) &= (X \wedge Y) \oplus (Y \wedge Z) \oplus (Z \wedge X); \\
\Sigma_0(X) &= ROTR^2(X) \oplus ROTR^{13}(X) \oplus ROTR^{22}(X); \\
\Sigma_1(X) &= ROTR^6(X) \oplus ROTR^{11}(X) \oplus ROTR^{25}(X).
\end{aligned}$$

The round function modifies the register as follows:

$$\begin{aligned}
T1_t &= H +_t \Sigma_1(E_t) + CH(E_t, F_t, G_t) + K_t + W_t;\\
T2_t &= \Sigma_0(A_t) + MJ(A_t, B_t, C_t);\\
H_{t+1} &= G_t; \quad G_{t+1} = F_t; \quad F_{t+1} = E_t; \quad E_{t+1} = D_t + T1_t;\\
D_{t+1} &= C_t; \quad C_{t+1} = B_t; \quad B_{t+1} = A_t; \quad A_{t+1} = T1_t + T2_t.
\end{aligned}$$

After all 64 input words have been input to the register, the resulting values of the state are added modulo 2^{32} to the initialized values of the state, according to the Davies-Meyer construction [17]. These values become the new intermediate hash value. If this is the last message block, the new intermediate hash value is output as the resulting message digest. Otherwise, the algorithm proceeds to updating the register using the next message block.

2.3 The Mundja CRC

Another component of Mundja, used to parallel the effect of the data expansion in SHA-256, is a 256-bit cyclic redundancy checksum of the message words, calculated over $GF(2^{256})$. This CRC is calculated word-at-a-time, using an eight word LFSR defined over $GF(2^{32})$. Binary Linear Feedback Shift Registers can be extremely inefficient in software on general-purpose microprocessors. LFSRs can operate over any finite field, so an LFSR can be made more efficient in software by utilizing a finite field more suited to the processor. Particularly good choices for such a field are the Galois Field with 2^w elements ($GF(2^w)$), where w is related to the size of items in the underlying processor, in this case 32-bit words. The elements of this field and the coefficients of the recurrence relation occupy exactly one unit of storage and can be efficiently manipulated in software.

The standard representation of an element A in the field $GF(2^w)$ is a w-bit word with bits $(a_{w-1}, a_{w-2}, \ldots, a_1, a_0)$, which represents the polynomial $a_{w-1}z^{w-1} + \ldots + a_1 z + a_0$. Elements can be added and multiplied: addition of elements in the field is equivalent to XOR. To multiply two elements of the field we multiply the corresponding polynomials modulo 2, and then reduce the resulting polynomial modulo a chosen irreducible polynomial of degree w.

It is also possible to represent $GF(2^w)$ using a subfield. For example, rather than representing elements of $GF(2^{31})$ as degree-31 polynomials over $GF(2)$, Mundja uses 8-bit octets to represent elements of a subfield $GF(2^8)$, and 32-bit words to represent degree-3 polynomials over $GF(2^8)$. This is isomorphic to the standard representation, but not identical. The subfield $B = GF(2^8)$ of octets is represented in Mundja modulo the irreducible polynomial $z^8 + z^6 + z^3 + z^2 + 1$. Octets represent degree-7 polynomials over $GF(2)$; the constant $\beta_0 = 0x67$ below represents the polynomial $z^6 + z^5 + z^2 + z + 1$ for example. The Galois finite field $W = B^4 = GF((2^8)^4)$ of words can now be represented using degree-3 polynomials where the coefficients are octets (subfield elements of B). For example, the word $0xD02B4367$ represents the polynomial $0xD0y3 + 0x2By2 + 0x43y + 0x67$. The field W can be represented modulo an irreducible polynomial $y^4 + \beta_3 y^3 + \beta_2 y^2 + \beta_1 y + \beta_0$.

The CRC polynomial defined over $GF(2^{32})$, then, is $x^8 + x^3 + \alpha$. The coefficient $\alpha \overset{\text{def}}{=} 0x00000100 = 0x00y^3 + 0x00y^2 + 0x01y + 0x00 = y$, was chosen because it allows an efficient software implementation: multiplication by α consists of retrieving a pre-computed constant from a table indexed by the most significant 8 bits, shifting the input word to the left by 8 bits of the multiplicand, and then adding (XORing) the resulting words together. This is essentially the field multiplication used in SNOW [3, 4] and is exactly that used in Turing [9] and SOBER-128 [10], so the same multiplication table can be used.

3 Description of Mundja

The message is presumed to be a sequence of octets. Four octets at a time are gathered (in little-endian fashion) to form a word. If necessary, a message that is not a multiple of four octets is treated as if it had extra trailing 0x00 octets, although the actual length is accounted for later.

Mundja requires input from the internal state of a secure stream cipher. This stream cipher may also be used to encrypt the message at the same time; the strength of Mundja does not rely on whether the message is encrypted or not.

Mundja consists of two registers: the first is a strengthened version of the SHA-256 register [5], with 8 words of state A, B, C, D, E, F, G, H; the second is a Cyclic Redundancy Checksum (CRC) with 8 words of state denoted $CRC[0], \ldots, CRC[7]$.

Padding: Suppose that the length of the message, M, is l octets. Append k zero octets 0x00, where k is the smallest, non-negative solution to the equation $l + k \equiv 0(\bmod 4)$. The length of the padded message should now be a multiple of 4 octets (that is, a multiple of 32-bits). The padded message must be less than 2^{64} words in length.

Parsing: The padded message is parsed into a sequence of 32-bit words, $\{M_t\}$.

Initialization: All stream ciphers are initialized using a secret key. It is well known that the keystream generated by a stream cipher should not be used more than once. In effect, this means that a stream cipher should start in a unique secret state for every message. For example, some stream ciphers require initialization using a *nonce*: a *n*umber used only *once*. These mechanism must also be applied when using Mundja. First, the stream cipher is initialized to obtain a unique secret state using the key and using whatever mechanisms are applicable to that stream cipher. Then, sufficient words of the stream cipher output or state are taken to initialize the two Mundja registers. The method used is dependent on the particular partner stream cipher: see appendices for some examples.

Register Update: Following initialization, the SHA round function and CRC update function are applied to accumulate the content of the padded and parsed message sequence $\{M_t\}$.

The round at which M_t is accumulated into the registers is the t-th round. The value of register state at the beginning of t-th round is denoted using a subscript t. The t-th round of the enhanced SHA round function modifies the SHA register also using input word W_t (from the partner stream cipher's current state) as input. The round function uses exactly the same functions as SHA-256 above, and additionally a nonlinear 32-bit function S. The function S is a non-linear function that may be based on the partner stream cipher's S-Box construction, if any (the appendices contain some examples). The authors advocate the use of the SOBER-128 f function (see [10]), as it is efficient and provides more than adequate resistance to attacks. The SHA round function modifies the SHA register according to the following algorithm:

$$\begin{aligned}
T1_t &= S(H_t + M_t + W_t) + \Sigma_1(E_t) + CH(E_t, F_t, G_t);\\
T2_t &= \Sigma_0(A_t) + MJ(A_t, B_t, C_t);\\
H_{t+1} &= G_t; \quad G_{t+1} = F_t; \quad F_{t+1} = E_t; \quad E_{t+1} = D_t + T1_t;\\
D_{t+1} &= C_t; \quad C_{t+1} = B_t; \quad B_{t+1} = A_t; \quad A_{t+1} = T1_t + T2_t.
\end{aligned}$$

The functions CH, MJ, Σ_0 and Σ_1 from SHA-256 are as specified above. The t-th input word is also input to the CRC register according to the following algorithm:

$$\begin{aligned} T_t &= M_t \oplus (\alpha \otimes CRC_t[0]) \oplus CRC_t[5]; \\ \text{for } j = 0..6: \quad & CRC_{t+1}[j] = CRC_t[j+1]; \\ CRC_{t+1}[7] &= T_t. \end{aligned}$$

Finalization and generation of MAC: When all the words of the input message have been processed, another word is mixed into the SHA register in the same manner as above. Note that whenever a word is input to the SHA register, the partner stream cipher state is updated as if a word of keystream has been generated. The input word is $0x6996c53a+2^{24}k$, recalling that k is the number of padding octets added. This word is **not** added to the CRC register. Adding this word to one register but not the other desynchronizes them in a manner that cannot be emulated by normal inputs, preventing extension attacks.

Now data from the CRC register is transferred into the SHA register, to emulate the effect of the data expansion in SHA-256. Eight times, the CRC register is cycled, and the value in $CRC[7]$ is input to the SHA register as above updating the stream cipher state as required to get W_i. This serves to further mix and propagate any differences generated in either register during the input phase.

Finally, to generate the MAC, this process (mixing values from the CRC register into the SHA register) is continued with the values of A (from the SHA register) after each mixing step used as the output MAC, until the requested amount of output has been produced. Words are converted to bytes in little-endian fashion, as usual for Mundja.

4 Analysis of Mundja

First, we consider the security properties required of a MAC function. A MAC function is a cryptographic algorithm that generates a tag $TAG = MAC_K(M)$ of length d from a message M of arbitrary length and a secret key K of length n. The message-tag pair (M, TAG) is transmitted to the receiver (the message may also be encrypted before transmission). Suppose the received message-tag pair is $(RM, RTAG)$. The receiver calculates an expected tag $XTAG = MAC_K(RM)$. If $XTAG = RTAG$, then the receiver has some confidence that the message-tag pair was formed by a party that knows the key K. In some cases, the message includes sequence data (such as a nonce) to prevent replay of message-tag pairs.

The length n of the key and the length d of the tag form the security parameters of a MAC algorithm, as these values dictate the degree to which the receiver can have confidence that the message-tag pair was formed by a party that knows the key K. A MAC function with security parameters (n, d) should provide resistance to four classes of attacks [17].

Collision Attack. In a collision attack, the attacker finds any two distinct messages M, M^* such that $MAC_K(M) = MAC_K(M^*)$. A MAC function resists a collision attack if the complexity of the attack is $O(2^{\min(n,d/2)})$.

First Pre-image Attack. In a first pre-image attack, the attacker is specified a tag value TAG, and the attacker must find a message M for which $MAC_K(M) = TAG$. A MAC function resists a first pre-image attack if the complexity of the attack is $O(2^{\min(n,d)})$.

Second pre-image attack In a second pre-image attack, the attacker is specified a message M, and the attacker generates a new message M^* such that $MAC_K(M) = MAC_K(M^*)$. A MAC function resists a second pre-image attack if the complexity of the attack is $O(2^{\min(n,d)})$.

MAC Forgery In MAC forgery, the attacker generates a new message-tag pair (M^*, y^*) such that $y^* = MAC_K(M^*)$. A MAC function resists MAC forgery if the complexity of the attack is $O(2^{\min(n,d)})$.

In all these attacks, the attacker is presumed to be ignorant of the value of the key K.[2] However, we assume that (prior to the attack) the attacker can specify messages $M^{(i)}$ for which they will be provided with the corresponding tags $TAG^{(i)} = MAC_K(M^{(i)})$.

Mundja is intended to be a MAC function with security parameters $n \leq 128$, and $d \leq 128$. That is, we claim that Mundja resists the above attacks when using 128-bit keys and outputting tags up to 128 bits in length. Naturally, these claims are based on the assumption that the accompanying stream cipher is secure.

Using this criteria, Mundja was designed with the following goals in mind.

Use 32-bit blocks. This minimizes the amount of padding and reduces the amount of computation. Mundja further assumes that messages consist of whole octets; that is, the message length is a multiple of 8 bits.

Utilize the SHA-256 round function. The cryptographers that designed SHA-256 (namely, the National Security Agency of the USA) are well respected, so using the SHA-256 round function seems a safe choice. The SHA hash algorithms [5] have two fundamental components: an almost-linear message schedule that expands the message to a longer sequence of input words; and the round function for combining the input words into a register.

Remove message expansion. A by-product of the SHA-256 message schedule is an asymptotic four-fold increase in the number of calls to the round function. Removing the message expansion increases the efficiency.

Strengthen the round function. The SHA-256 round function is strong, although not as strong as the public anticipated [11]. We chose to use a strengthened version of the SHA-256 round function. A problem with the SHA-256 round function is that the carries in the modular addition distribute non-linear information to only a few bits. The most effective way to strengthen the round function is to include a non-linear function S (such as an 8-bit input S-box). This distributes non-linear information amongst a large number of bits (when compared to modular addition). We considered various options for combining a nonlinear function into the SHA round function. The best resistance to differential collisions was observed when $(H_t + M_t + W_t)$ is replaced with $S(H_t + M_t + W_t)$ when computing $T1_t$. The round function is otherwise unchanged.

Accumulate on a second register (CRC register). The round function is weak when considered on its own [11], but the combination with the message data expansion provides significantly more security. Since Mundja has no message expansion, Mundja must include some other method for preventing attacks. We decided to accumulate the message in a second orthogonal register for which any attempt to cancel differences in the SHA register

[2] There are circumstances (albeit rare) where the users require a MAC function to resist a collision attack with *known* key. Mundja is not intended to prevent this type of attack. We note that other common MAC constructions, such as CBC-MAC [12], cannot prevent this type of attack either.

will introduce complex differences in this second register (and vice versa). We have chosen to use a $GF(2^{32})$ CRC as the second register. The properties of such CRCs are well understood and thus the CRC allows more concrete analysis.

Input existing stream cipher internal state to the SHA register. The internal state of the stream cipher is already computed, and is unknown to a hypothetical attacker, so it makes sense to utilize this secret data. For Mundja, the SHA-256 round constants are replaced by inputs from the stream cipher internal state. This introduces key-dependent material at each round, thus re-randomizing the register state. □

Security Claims. We claim that, with security parameters $n \leq 128$, and $d \leq 128$, Mundja resists the four classes of attacks mentioned above. The best collision attacks are expected to be based on forming differentials in the SHA register and CRC register before finalization. Such differential collisions are considered in detail in Section 5. It is also possible that the values in the SHA register and CRC register after accumulating the first message are different from the values in the SHA register and CRC register after accumulating the first message, with the finalization step cancelling these differences. Research in this direction is ongoing.

A first pre-image attack would require reversing the SHA round algorithm, so such attacks seems unlikely. The best second pre-image attacks and MAC forgery attacks are expected to be based on forming differential collisions.

Our hope is that, with a strong S function, Mundja can resist all attacks even when using 256-bit keys. This would require an accompanying stream cipher that is designed for 256-bit keys. We have not yet ascertained if Mundja can offer this level of security.

5 Differential Analysis of Mundja

Our analysis of Mundja is related to the analysis of SHA-2 family.

Corrective Patterns of SHA-256. There are two prior publications analyzing SHA-256: an analysis by Gilbert and Handschuh [7]; and an analysis by Hawkes, Paddon and Rose [11]. Thus far, the most successful analysis of SHA algorithms resulted from the search for Chaboud-Joux differential collisions [1, 2, 7, 15, 20]. This type of analysis first finds a high-probability corrective pattern for the round function; that is, given one sequence of inputs to the round function, the analysis finds another sequences of inputs such that both sequences of inputs result in identical register states. The analysis of the message schedule looks for two messages such that the message schedule will result in high-probability differential collisions being input to the register.

The approach of [11] considers both XOR-differences $\varDelta X = X^* \oplus X$ and addition-differences $\delta X = X^* - X \pmod{2^{32}}$: addition-differences are so-called because the differences are formed relative to the modular addition group operation. Both differences are useful. An analysis of the $\varSigma$ functions with respect to XOR-differences is simple, while analysis of $\varSigma$ functions with respect to addition-differences is quite complex. However, the remainder of the round function is best analyzed using addition-differences. As noted in [11], if $\varDelta X = \lambda$, then δX can be determined if $X[i]$ is known for every $i < 31$ such that $\lambda[i] = 1$.[3] That is, if the attacker predicts the bits of X at the positions where $\widehat{\lambda}[i] = 1$, then the attacker also predicts δX.

[3] The attacker need not guess $X[31]$ to determine δX, since differences in the most significant bit always contribute an addition-difference of 2^{31}.

j		H	G	F	E	D	C	B	A	δM_{t+j}	δS^{out}_{t+j}	$\#S$	Ass.	Gue.
			CH inputs				MJ inputs							
0	Δ	-	-	-	-	-	-	-	-	δS^{in}_0	$\delta E_1/\delta A_1 = \delta_0$	$\widehat{\alpha 1}$	$\widehat{\alpha 1}$	$\widehat{\alpha 1}$
	δ	-	-	-	-	-	-	-	-		$\Delta E_1/\Delta A_1 = \alpha 1$			
1	Δ	-	-	-	$\alpha 1$	-	-	-	$\alpha 1$	δS^{in}_1	$-\delta\Sigma_1(E_1)$	$\widehat{\beta 3}$	$\alpha 1$,$\alpha 1$	$\widehat{\beta 3}$
	δ	-	-	-	δ_0	-	-	-	δ_0		$-\delta\Sigma_0(A_1)$	$\widehat{\gamma 3}$	$\widehat{\beta 3}$	$\widehat{\gamma 3}$
										(Write $\delta\Sigma_0(A_1) = \delta_1$)				
2	Δ	-	-	$\alpha 1$	$\beta 3$	-	-	$\alpha 1$	-	δS^{in}_2	$-\delta\Sigma_1(E_2)$	$\widehat{\epsilon 9}$	$\alpha 1$,$\alpha 1$	$\widehat{\epsilon 9}$
	δ	-	-	δ_0	$-\delta_1$	-	-	δ_0	-				$\beta 3$	
3	Δ	-	$\alpha 1$	$\beta 3$	-	-	$\alpha 1$	-	-	-	-	-	$\alpha 1$,$\alpha 1$	-
	δ	-	δ_0	$-\delta_1$	-	-	δ_0	-	-				$\beta 3$	
4	Δ	$\alpha 1$	$\beta 3$	-	-	$\alpha 1$	-	-	-	$-\delta H_4 = -\delta_0$	-	-	$\beta 3$	-
	δ	δ_0	$-\delta_1$	-	-	δ_0	-	-	-				$\widehat{\alpha 1}$	
5	Δ	$\beta 3$	-	-	$\alpha 1$	-	-	-	-	$-\delta H_5 + \delta S^{in}_5$	$-\delta\Sigma_1(E_5)$	$\widehat{\gamma 3}$	$\alpha 1$	$\widehat{\gamma 3}$
	δ	$-\delta_1$	-	-	δ_0	-	-	-	-	$= \delta_{1,2} + \delta S^{in}_5$				
6	Δ	-	-	$\alpha 1$	-	-	-	-	-	-	-	-	$\alpha 1$	-
	δ	-	-	δ_0	-	-	-	-	-					
7	Δ	-	$\alpha 1$	-	-	-	-	-	-	-	-	-	$\alpha 1$	-
	δ	-	δ_0	-	-	-	-	-	-					
8	Δ	$\alpha 1$	-	-	-	-	-	-	-	$-\delta H_8 = -\delta_0$	-	-	-	-
	δ	δ_0	-	-	-	-	-	-	-					

Table 1. A summary of the corrective patterns for Mundja. The differences shown round j are the values before the round function is applied.

The corrective pattern is successful when certain conditions on the internal state are satisfied, and when the attacker correctly guesses certain bits of the internal state. When the single bit difference $\alpha 1 = 2^{31}$ is used, then the conditions have probability 2^{-9} of being correct and the attacker must guess 30 bits of internal state. Thus, if the internal state is unknown, then the probability of success is 2^{-39}.

Corrective Patterns for the Mundja Round Function. The high probability of the SHA-256 corrective pattern exists because of the prevalent use modular addition operations. Thus, addition-differences in the input word W_t can be used to cancel any addition-difference(s) in any values being fed back. As we shall see, the inclusion of a nonlinear function S in Mundja prevents an attacker from simply providing message words with the correct addition-difference: now the attack must provide message words for which the *output* of S has the correct addition-difference. This complicates attacks significantly.

To analyze the Mundja SHA register, we must divide the evaluation of $T1_t$ into three separate evaluations:

$$S^{in}_t = H_t + M_t + W_t; \quad S^{out}_t \;=\; S(S^{in}_t);$$
$$T1_t = S^{out}_t + \Sigma_1(E_t) + CH(E_t, F_t, G_t).$$

Table 1 summarizes the corrective pattern for the Mundja SHA register. This corrective pattern is based on the SHA-2 corrective pattern of [11]. In this table, $\alpha 1$ represents a single-bit XOR-difference, with $\beta 3 = \Sigma_0(\alpha 1)$, $\gamma 3 = \Sigma_1(\alpha 1)$ and $\epsilon 9 = \Sigma_1(\Sigma_0(\alpha 1)) = \Sigma_1(\beta 3)$. The number after the Greek letter indicates the Hamming weight of the difference. For each round j, the rows beginning with Δ and δ list the XOR-differences and addition-differences respectively. The δM_{t+j} column lists the addition-differences required for the corrective pattern. Note that values such as H_{t+4} have been written H_4 to save space. The δS_{t+j}^{out} column lists the required addition-differences in the output of the S function. The final three columns provide the data for determining the probability of the corrective pattern. The column headed by "$\#S$" indicates the bit positions in the output of S where the attacker will have to generate the correct bits to cause cancellation: the attacker will have to generate the correct bits at the positions where the bits are "1" in the listed value(s). This affects the choice of δS_{t+j}^{in}, noting that δS_{t+j}^{in} will depend on the construction of S. The column headed by "Ass." indicates the bit positions where the attacker must assume conditions on register states, while the last column (headed by "Gue.") indicates the bit positions where the attacker must guess bits.

Round 0 Inject a small XOR-difference $\alpha 1$ into E_1 and A_1.

- Guess E_1 at the bits where $\widehat{\alpha 1}[i] = 1$, to determine δS_0^{out} that gives $\Delta E_1 = \alpha 1$. Add $\widehat{\alpha 1}$ to Gue..
- • NOTE: attacker requires $\delta S_0^{out} = \delta E_1 \overset{\text{def}}{=} \delta_0$ to get $\Delta E_1 = \alpha 1$. Implies $\delta A_1 = \delta_0$.
- Make assumptions on A_1 at bits where $\widehat{\alpha 1}[i] = 1$, to ensure that $\Delta A_1 = \alpha 1$. Add $\widehat{\alpha 1}$ to Ass.
- To obtain correct $\delta S_0^{out} = \delta_0$, the attacker controls S_0^{out} at $\widehat{\alpha 1}$. Add $\widehat{\alpha 1}$ to $\#S$.
- The attacker inserts $\delta M_t = \delta S_0^{in}$.

Round 1 Cancel differences in Σ_0 and Σ_1. Assume $\delta CH_1 = \delta MJ_1 = 0$.

- • NOTE: at bits where $\Delta E_1[i] = 1$, CH chooses $F_1[i]$ for one message and $G_1[i]$ for the other message.
- Make assumptions $F_1[i] = G_1[i]$ at bits where $\Delta E_1[i] = \alpha 1[i] = 1$, to ensure that no difference in CH results. Add $\alpha 1$ to Ass..
- Make assumption $B_1[i] = C_1[i]$ at bits where $\Delta A_1[i] = \alpha 1[i] = 1$, to ensure that no difference in MJ results. Add $\alpha 1$ to Ass..
- • NOTE: $\Delta E_1 = \alpha 1$ implies $\Delta \Sigma_1(E_1) = \gamma 3$.
- Guess $\Sigma_1(E_1)$ at bits where $\widehat{\gamma 3}[i] = 1$, to determine $\delta \Sigma_1(E_1)$. Gue. $\widehat{\gamma 3}$.
- • NOTE:$\Delta A_1 = \alpha 1$ implies $\Delta \Sigma_0(A_1) = \beta 3$.
- Guess $\Sigma_0(A_1)$ at bits where $\widehat{\beta 3}[i] = 1$, to determine $\delta \Sigma_0(A_1)$. Gue. $\widehat{\beta 3}$.
- Attacker requires $\delta S_1^{out} = -\Sigma_1(E_1) - \Sigma_0(A_1)$ to get $\delta A_2 = 0$. This implies that $\delta E_2 = \delta S_1^{out} + \Sigma_1(E_1) = -\Sigma_0(A_1) \overset{\text{def}}{=} -\delta_1$.
- Make assumptions E_2 at bits where $\widehat{\beta 3}[i] = 1$, to ensure that $\delta E_2 = -\Sigma_0(A_1) = -\delta_1$, results in $\Delta E_2 = \beta 3$. Add $\widehat{\beta 3}$ to Ass..
- To obtain correct $\delta S_1^{out} = -\Sigma_1(E_1) - \Sigma_0(A_1)$, the attacker must control S_1^{out} at bits where $\widehat{\gamma 3}[i] = 1$, and the bits where $\widehat{\beta 3}[i] = 1$. Add $\widehat{\gamma 3}, \widehat{\beta 3}$ to $\#S$.
- The attacker inserts $\delta M_{t+1} = \delta S_1^{in}$.

Round 2 Cancel differences in Σ_1. Assume $\delta CH_2 = \delta MJ_2 = 0$.

- • NOTE: at bits where $\Delta E_2[i] = 1$, CH_2 chooses $F_2[i]$ for one message and $G_2[i]$ for the other message.
- Make assumptions on G_2 and F_2 at bits where $\Delta E_2[i] = \beta 3[i] = 1$, to ensure that no difference in CH results. Add $\beta 3$ to Ass..
- • NOTE: at bits where $\Delta F_2[i] = 1$, selector $E_2[i]$ chooses $G_2[i]$ to get $\Delta CH_2[i] = 0$.
- Make assumptions on E_2 at bits where $\Delta F_2[i] = \alpha 1[i] = 1$, to ensure that no difference in CH results. Add $\alpha 1$ to Ass..
- Make assumption $A_2[i] = C_2[i]$ at bits where $\Delta B_2[i] = \alpha 1[i] = 1$, to ensure that no difference in MJ results. Add $\alpha 1$ to Ass.

- NOTE: $\Delta E_2 = \beta 3$ implies $\Delta \Sigma_1(E_2) = \epsilon 9$.
  - Guess $\Sigma_1(E_1)$ at bits where $\widehat{\epsilon 9}[i] = 1$, to determine $\delta \Sigma_1(E_2)$. Gue. $\widehat{\epsilon 9}$.
  - Attacker requires $\delta S_2^{out} = -\Sigma_1(E_2)$, to get $\delta A_3 = 0$.
  - To obtain correct $\delta S_2^{out} = -\delta \Sigma_1(E_2)$, the attacker must control S_2^{out} at bits where $\widehat{\epsilon 9}[i] = 1$. Add $\widehat{\epsilon 9}$ to $\#S$.
  - The attacker inserts $\delta M_{t+2} = \delta S_2^{in}$.

Round 3 Assume $\delta CH_3 = \delta MJ_3 = 0$.

- NOTE: at bits where $\Delta F_3[i] = 1$, selector $E_3[i]$ chooses $G_3[i]$ to get $\Delta CH_3[i] = 0$.
  - Make assumptions on E_3 at bits where $\beta 3[i] = 1$, to ensure that no difference in CH results from $\Delta F_3 = \beta 3$. Add $\beta 3$ to Ass..
- NOTE: where $\Delta G_3[i] = 1$, selector $E_3[i]$ must choose $F_3[i]$ to ensure $\Delta CH_3[i] = 0$.
  - Make assumptions on E_3 at bits where $\alpha 1[i] = 1$, to ensure that no difference in CH results from $\Delta G_3 = \alpha 1$. Add $\alpha 1$ to Ass..
  - Make assumption $A_3[i] = B_3[i]$ at bits where $\Delta C_3[i] \alpha 1[i] = 1$, to ensure that no difference in MJ results. Add $\alpha 1$ to Ass..
- NOTE: $\delta S_3^{out} = 0$ and $\#S = 0$.
  - The attacker inserts $\delta M_{t+3} = 0$.

Round 4 Cancel difference in H_4. Assume $\delta CH_4 = 0$.

- The value $\delta H_4 = \delta E_1$, is known by the attacker, so no guess required.
- Make assumptions on E_4 at bits where $\beta 3[i] = 1$, to ensure that no difference in CH results from $\Delta G_4 = \beta 3$. Add $\beta 3$ to Ass..
- Attacker requires $\delta S_4^{out} = 0$, to get $\delta A_5 = 0$. Thus $\#S = 0$.
- NOTE: $\delta T1_4 = 0$, and thus $\delta E_5 = \delta D_4 = \delta_0$, where $\Delta D_4 = \alpha$.
  - Make assumptions on E_5 at the bits where $\Delta D_4[i] = \alpha 1[i] = 1$, to ensure that the single bit difference in D_4 becomes a single bit difference $\alpha 1$ in E_5. Ass. $\alpha 1$.
  - To cancel δH_4, the attacker inserts $\delta M_{t+4} = \delta H_4$.
  - NOTE: The value of $\delta H_4 = -\delta_1$, is known by the attacker, so no guess required.

Round 5 Cancel differences in H_5 and Σ_1. Assume $\delta CH_5 = 0$.

- Make assumptions $F_5[i] = G_5[i]$ at bits where $\Delta E_5[i] = \alpha 1[i] = 1$, to ensure that no difference in CH results. Add $\alpha 1$ to Ass..
- NOTE: $\Delta E_5 = \alpha 1$ implies $\Delta \Sigma_1(E_5) = \gamma 3$.
  - Guess the value of $\Sigma_1(E_5)$ at the bits where $\widehat{\gamma 3}[i] = 1$, to determine $\delta \Sigma_1(E_5)$. Add $\widehat{\gamma 3}$ to Gue..
  - To obtain correct δS_5^{out}, the attacker must know S_5^{out} at bits where $\widehat{\gamma 3}[i] = 1$. Add $\widehat{\gamma 3}$ to $\#S$.
  - To cancel δH_5, the attacker inserts $\delta M_{t+5} = -\delta H_5$.
  - NOTE: The value of $\delta H_5 = -\delta_1$, is known by the attacker, so no guess required.
  - Attacker sets $\delta M_{t+5} = \delta S_5^{in} - \delta H_5 = \delta S_5^{in} + \delta_1$.

Round 6 Assume $\delta CH_6 = 0$.

- Make assumptions on E_6 at bits where $\alpha 1[i] = 1$, to ensure that no difference in CH results from $\Delta F_6 = \alpha 1$. Add $\alpha 1$ to Ass..
- Attacker requires $\delta S_6^{out} = 0$, to get $\delta A_7 = 0$. Thus $\#S = 0$.
- Attacker sets $\delta M_{t+6} = 0$.

Round 7 Assume $\delta CH_7 = 0$.

- Make assumptions on E_7 at bits where $\Delta G_7[i] = \alpha 1[i] = 1$, to ensure that no difference in CH results. Add $\alpha 1$ to Ass..
- Attacker requires $\delta S_7^{out} = 0$, to get $\delta A_8 = 0$. Thus $\#S = 0$.
- Attacker sets $\delta M_{t+7} = 0$.

Round 8 Cancel difference in H_8..
- Attacker requires $\delta S_8^{out} = 0$, to get $\delta A_9 = 0$. Thus $\#S = 0$.
- To cancel δH, the attacker injects $\delta M_{t+8} = -\delta H_8$.
- NOTE: The value of $\delta H_8 = \delta_0$ is known by the attacker, so no guess required.
- Attacker sets $\delta M_{t+8} = -\delta H_8 = -\delta_0$.

Probability of the Corrective Pattern. From Table 1 we see that the total number of bits for which the attacker makes assumptions is $(9|\alpha 1| + 2|\widehat{\alpha 1}| + 3|\beta 3| + 1|\widehat{\beta 3}|)$. This value attains a minimum of 21 when $\alpha 1 = 0x80000000$. The maximum value is 23. The total number of bits that are guessed by the attacker is $(1|\widehat{\alpha 1}| + 1|\widehat{\beta 3}| + 2|\widehat{\gamma 3}| + 1|\widehat{\epsilon 9}|)$. When $\alpha 1 = 0x80000000$, the attacker guesses 18 bits of state. Thus, the assumptions are correct and a guesses are correct with probability 2^{-39}. The minimum probability resulting from a single bit difference is 2^{-42}. However, we have yet to account for the non-linear function S in rounds 0, 1, 2, and 5. The attacker needs to determine the suitable difference δS^{in} to obtain the correct difference δS^{out}. This then allows the attacker to determine the correct values of δM_{t+j}.

We consider the simplest case where S is a SOBER-128 f function [10] wherein the 8 most significant bits become input to an 8×32 S-box and the 32-bit S-box output is XORed with the input to obtain the output of f. For such an S function, the output differences ΔS^{out} can be classified into two groups according to whether ΔS^{out} has any bit difference in the 8 MSBs, or not.

- If ΔS^{out} has no bit differences in the 8 MSBs, then the attacker must only guess the output of the S-box at the bits positions where $\Delta S^{out}[i] = 1$ in order to determine S^{in} from S^{out}. The attacker can then determine the δS^{in} and thus choose δM_{t+j}. This adds a factor of 2^{-1} to the probability for each bit difference in ΔS^{out}.
- If ΔS^{out} has any bit difference in the 8 MSBs, then the attacker must guess all 8 MSBs of S^{in} to get the correct difference in the 8 MSBs of S^{out}. The attacker must also introduce differences in the message to cancel out the differences that result in the lower 24 bits of the output of the S-box. As a simple lower bound, it is expected that the attacker will have to correct at least two such bit difference to obtain the correct difference in the lower 24 bits of S^{out}. That is, the probability of getting the correct difference in the lower 24 bits is 2^{-2}. Thus, whenever ΔS^{out} has any bit difference in the 8 MSBs, then this is expected to add a total factor of $2^{-8} \times 2^{-2} = 2^{-10}$ to the probability.

Table 2 looks at the probabilities that result when accounting for the S-box in rounds 0, 1, 2 and 5. The last two columns compare the probabilities that result when using $\alpha 1 = 2^{31} = 0x80000000$ and $\alpha 1 = 2^{13} = 0x00002000$ (which maximizes the probability). For these two cases, the last three rows show: the probability from the S-box; the probability from the assumptions and guesses; and the total probability of the corrective pattern.

5.1 Corrective Patterns and the Mundja CRC

The value of $CRC_{t+1}[7]$ may be expressed as:

$$CRC_{t+1}[7] = \bigoplus_{i=0}^{t} X[t-i] \otimes M_i \quad \oplus \quad \bigoplus_{j=0}^{7} Y[j,t] \otimes CRC_0[j],$$
$$\text{where } 0 = X[a+8] \oplus X[a+5] \oplus (\alpha \otimes X[a]), \tag{1}$$

Round	ΔS^{out}_{t+j}	Probability when:			
		Diff. in 8 MSBs	No Diff. in 8 MSBs	$\alpha 1 = 2^{31}$ $0x80000000$	$\alpha 1 = 2^{13}$ $0x00002000$
0	$\alpha 1$	2^{-10}	2^{-1}	2^{-10}	2^{-1}
1	$\beta 3 \oplus \gamma 3$		2^{-6}	2^{-10}	2^{-6}
2	$\epsilon 9$		2^{-9}	2^{-10}	2^{-10}
5	$\gamma 3$		2^{-3}	2^{-10}	2^{-3}
S-box Contribution				2^{-40}	2^{-20}
Probability for Ass. and Gue.				2^{-39}	2^{-42}
Total Probability of Corrective Pattern				2^{-79}	2^{-62}

Table 2. Probabilities resulting from the S-box.

and $Y[j,t] \in GF(2^{32})$ are constants. Now suppose we input messages M and M^* with $\Delta M_i = (M_i^* \oplus M_i)$. The difference $\Delta CRC_{t+1}[7] = CRC^*_{t+1}[7] \oplus CRC_{t+1}[7]$ between the CRC states resulting from M and M^* satisfy:

$$\Delta CRC_{t+1}[7] = \bigoplus_{i=0}^{t} X[t-i] \otimes \Delta M_i.$$

The sequence $\{X[a]\}$ satisfies the recurrence $0 = X[a+8] \oplus X[a+5] \oplus (\alpha \otimes X[a])$. There are many linear relationships between elements in this sequence; some more useful linear relationships correspond to repeated squaring of the characteristic polynomial:

$$\begin{aligned} 0 &= X[a+2^i \cdot 8] \oplus X[a+2^i \cdot 5] \oplus (\alpha^{2^i} \otimes X[a]), \\ \Rightarrow 0 &= X[a+2^{32} \cdot 8] \oplus X[a+2^{32} \cdot 5] \oplus (\alpha \otimes X[a]), \end{aligned} \qquad (2)$$

since $\gamma^{2^{32}} = \gamma$, $\forall \gamma \in GF(2^{32})$. The XOR of Equations (1) and (2) results in

$$\begin{aligned} 0 &= X[a+2^{32} \cdot 8] \oplus X[a+2^{32} \cdot 5] \oplus X[a+8] \oplus X[a+5], \\ \Rightarrow 0 &= X[a+2^{32} \cdot 8 - 5] \oplus X[a+2^{32} \cdot 5 - 5] \oplus X[a+3] \oplus X[a]. \end{aligned}$$

Note that if we define constants

$$\begin{aligned} d_4 &= (2^{32} \cdot 8 - 5), & d_3 &= d_4 - 3 \;=\; 2^{32} \cdot 8 - 8, \\ d_2 &= d_4 - (2^{32} \cdot 5 - 5) \;=\; 2^{32} \cdot 3, & d_1 &= d_4 - (2^{32} \cdot 8 - 5) \;=\; 0, \end{aligned}$$

then the sequence $\{X[a]\}$ satisfies $\oplus_{k=1}^{4} X[t-d_k] = 0$, for any value of $t \geq d_4$. Suppose that there is a corrective pattern with XOR differences: $(\Delta M[i], \ldots, \Delta M[i+l]) = (\Delta_0, \ldots, \Delta_l)$. If the message M is of sufficient length, then an attacker can input differences (according to the corrective pattern) at four positions: $a_i = a + d_i$, where d_1, d_2, d_3, d_4 are as defined above. Then, for all $t \geq a + d_4$:

$$\Delta CRC_{t+1}[7] = \bigoplus_{i=0}^{t} (X[t-i] \otimes \Delta M_i) = \bigoplus_{j=0}^{l} \left(\bigoplus_{k=1}^{4} X[(t-a+j) - d_k] \right) \otimes \Delta_j = 0,$$

upon substitution of $\oplus_{k=1}^{4} X[(t-a+j) - d_k] = 0$. That is, if we input the differences at these four places a_1, a_2, a_3, a_4, then the resulting differences in the CRC state cancel out for all $t > a_4$. In this way, the attacker has forced a collision in the CRC without guessing any of the CRC register. Additionally, the corrective patterns cancel out the differences in the SHA register.

Collision Probability. The probability of the four corrective patterns cancelling out the differences in the SHA register is anticipated to approach p_{CP}^4, where p_{CP} is the probability of the corrective pattern, since the four corrective patterns are largely independent. The attacker can use the optimal corrective pattern (that is the corrective pattern with highest complexity) at each of the four positions. Substituting $p_{CP} = 2^{-62}$ from Table 2 gives a probability of 2^{-248}. In other words, the complexity of forming a collision is 2^{260}. This is significantly more than the complexity of 2^{-128} required for Mundja to be secure.

We have considered whether there is an advantage to be gained from using other linear relationships to cancel differences in the CRC. Thus far we have been unable to find any useful linear relationships. This work is ongoing.

6 Conclusion

Mundja is a MAC function designed for use with stream ciphers. Mundja uses a strengthened version of the SHA-256 round function and a 256-bit cyclic redundancy check (CRC). The design philosophy is presented, along with an analysis of resistance to differential collisions. This supports our claim that Mundja offers up to 128-bit security when used with a secure stream cipher.

References

1. Eli Biham, Rafi Chen, *New results on SHA-0 and SHA-1,* Short talk presented at CRYPTO 2004 Rump Session, 2004.
2. F. Chabaud and A. Joux, *Differential Collisions in SHA-0,* Advances in Cryptology-CRYPTO'98, Lecture Notes in Computer Science, vol.1462, pp.56-71, Springer-Verlag, 1998.
3. P. Ekdahl and T. Johansson, *SNOW - a new stream cipher,* 2000. This paper can be found in the NESSIE web pages: http://ww.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions/snow.zip.
4. P. Ekdahl and T. Johansson, *SNOW - a new stream cipher,* Selected Areas in Cryptography, SAC 2002, St. John's, Canada, Lecture Notes in Computer Science, vol. 2595, K. Nyberg and H. Heys (Eds.), pp. 47-61, Springer, 2002.
5. National Institute of Standards and Technology, *Federal Information Processing Standards (FIPS) Publication 180-2, Secure Hash Standard (SHS),* February, 2004.
6. S. Fluhrer and D. McGrew, *Statistical Analysis of the Alleged RC4 Keystream Generator,* Fast Software Encryption, FSE 2000, New York, USA, Lecture Notes in Computer Science, vol. 1978, B. Schneier (Ed.), pp. 19-30, Springer, 2004.
7. H. Gilbert and H. Hanschuh, *Security Analysis of SHA-256 and sisters,* Selected Areas in Cryptography, SAC 2003, Canada, Lecture Notes in Computer Science, vol. 3006, M. Matsui and R. Zuccheratopp (Eds.), pp. 175-193, Springer, 2004.
8. K. Gutpa and S. Maitra, *Multiples of primitive polynomials over GF(2),* Progress in Cryptology -INDOCRYPT2001, Lecture Notes in Computer Science, vol. 2247, C. Rangan and C. Ding (Eds.), pp. 62-72, Springer, 2001.
9. P. Hawkes and G. Rose, *Turing, a fast stream cipher,* Cryptology ePrint Archive, Report 2002/185, see `http://eprint.iacr.org/`, 2002.
10. P. Hawkes and G. Rose, *Primitive Specification for SOBER-128,* Cryptology ePrint Archive, Report 2003/081, see `http://eprint.iacr.org/`, 2003.

11. P. Hawkes, M. Paddon and G. Rose, *On Corrective Patterns for the SHA-2 Family,* Cryptology ePrint Archive, Report 2004/207, see `http://eprint.iacr.org/`, 2004.
12. International Organization for Standardization, *Data Cryptographic Techniques-Data Integrity Mechanism Using a Cryptographic Check Function Employing a Block Cipher Algorithm,* ISO/IEC 9797, 1989.
13. A. Joux and F. Muller, *A Chosen IV Attack Against Turing,* Selected Areas in Cryptography, SAC 2003, Ottawa, Canada, Lecture Notes in Computer Science, vol. 3006, M. Matsui and R. Zuccheratopp (Eds.), pp. 194-207, Springer, 2004.
14. A. Joux, *Multicollisions in Iterated Hash Functions,* Advances in Cryptology - CRYPTO 2004, Lecture Notes in Computer Science, vol. 3152, M. Franklin (Ed.), pp. 306-316, Springer, 2004.
15. A. Joux, *Collisions in SHA-0,* Short talk presented at CRYPTO 2004 Rump Session, 2004.
16. H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication,* Internet RFC 2104, February 1997.
17. A. Menezes, P.van Oorschot and A. Vanstone, *Handbook of Applied Cryptography,* CRC Press series on Discrete Nathematics and its Applications, CRC Press LLC, 1997.
18. M. Mihaljevic and H. Imai, *Cryptanalysis of Toyocrypt-HS1 stream cipher,* IEICE Transactions on Fundamentals, vol E85-A, pp. 66-73, January 2002. Available at www.csl.sony.co.jp/ATL/papers/IEICEjan02.pdf.
19. L. Simpson, E. Dawson, J. Golic and W. Millan, *LILI Keystream Generator;* Selected Areas in Cryptography SAC'2000, Lecture Notes in Computer Science, vol. 1807, D. Stinson and S. Tavares (Eds.), pp. 392-407, Springer, 2000.
20. X. Wang, D. Feng, X. Lai and H. Yu, *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD,* Cryptology ePrint Archive, Report 2004/199, see `http://eprint.iacr.org/`, 2004.

A Appendix: How Mundja cooperates with partner stream ciphers

The description of Mundja above makes reference to "the partner stream cipher". Some aspects of Mundja (the CRC multiplication table and the constant value used in finalization) have been borrowed from SOBER-128, and are used no matter what the partner stream cipher is. The following elements of the design differ depending on the partner:

– The manner in which the SHA and CRC registers are initialized after stream cipher initialization.
– The stream of words W_t used in place of the SHA-256 round constants.
– The nonlinear S-box used to strengthen the SHA-256 round function. Some stream ciphers might not have any equivalent of an S-box available, in which case we believe that the SOBER-128 S-box construction (as analyzed above) should be used.

We define these missing elements for three ciphers: SOBER-128, Turing, and RC4. Reference source code for Mundja partnering all three of these ciphers will be available on the QUALCOMM Australia web site:
`http://www.qualcomm.com.au`.

A.1 SOBER-128

SOBER-128 already supports initialization using a secret key and a nonce. After initialization, the SOBER-128 register words $R[i], i = 0..15$ are simply copied into the SHA register words $A, B, \ldots, H$ and the CRC register words $CRC[j], j = 0..7$, respectively.

Each time the SHA register is updated, a word from the SOBER-128 LFSR $R[8]$ is taken as W_t.

The SOBER-128 f function is used as the function S in Mundja. Note that this function uses an 8×32 S-box (denoted $SBOX$) with f defined as $f(X) = SBOX(X >> 24) \oplus X$.

A.2 Turing

Turing already supports initialization using a secret key and a nonce, however Joux and Miller [13] observed that there was insufficient diffusion during this process. So, after initialization, Turing is used to generate 20 words of stream cipher output (recall that Turing generates keystream in 5-word chunks). The first 16 of these words are simply copied into the SHA register words $A, B, \ldots, H$ and the CRC register words $CRC[j], j = 0..7$, respectively. Note that this process provides sufficient extra diffusion in the Turing LFSR to defeat the [13] attack.

Each time the Turing LFSR is updated, a word from the LFSR $R[8]$ is taken as W_t. The Turing *Sbox* function is used as the S-box. this S-box is significantly better than the SOBER-128 one, in that it is highly nonlinear in all of the input bits, and it is secret-key dependent. We expect that this adds significant strength to Mundja when used in conjunction with Turing.

A.3 RC4

We do not recommend use of RC4 as a stream cipher any more, principally due to the strong distinguishing attack of Fluhrer & McGrew [6]. Nevertheless, we feel it is instructive to specify how Mundja would partner with RC4.

RC4 does not directly support use of a nonce, and yet this is a requirement for use with Mundja. So, following Ron Rivest's advice, first initialize the RC4 state with a key that is the MD5 hash of the secret key and nonce concatenated. Then 64 octets of RC4 keystream are generated, and formed into 16 32-bit words in little-endian fashion. These words are simply copied into the SHA register words $A, B, \ldots, H$ and the CRC register words $CRC[j], j = 0..7$, respectively.

RC4 has an index variable i, and a state permutation $S[]$. It generates stream output octet-at-a-time, whereas Mundja works word-at-a-time. Synchronization between these processes needs to be clarified, so we choose to have RC4 generate 4 octets of output before Mundja accumulates its input word. For each Mundja input word, the four octets of S that (cyclically) precede the current index i (i.e. the four octets that have just been swapped) are formed into a word in little-endian fashion and taken as W_t.

RC4 does not have a particular non-linear S-box, but the current state S forms an 8×8 permutation and can naturally be used to provide this functionality. Where Mundja needs a 32-bit S-box, the input is broken into octets, and each octet is individually passed through the S permutation (in parallel) before being reassembled into a word.