Building the Big Message Authentication Code

Dale R. THOMPSON, J. Brad MAXWELL, and James P. PARKERSON
Computer Science and Computer Engineering, University of Arkansas
Fayetteville, Arkansas 72701, U.S.A.

ABSTRACT

Message Authentication Codes (MAC) are used to ensure the integrity of digital content is not compromised and to authenticate the sender. A MAC can be constructed by using a message digest and a shared secret value. Two different digital contents should have a low probability of having the same message digest. This is called a collision. Also, when a message digest is used to create a MAC, it should be difficult for an adversary to create a collision. A rough calculation demonstrates that a collision could be found for a 128-bit message digest in approximately six hours with a parallel collision attack using a special-purpose machine. A composite message digest is constructed out of existing functions to decrease the probability of collision. Also, the MAC based on the composite message digest is constructed so that it is difficult for an adversary to create a collision.

Keywords: Authentication, Message authentication codes, and message digest.

1. INTRODUCTION

Message authentication permits two or more parties to verify that the received digital content is genuine [1]. Assuring the authenticity of digital content is important for electronic communication such as e-mail, e-commerce, and a paperless environment. A Message Authentication Code (MAC), sometimes called a cryptographic checksum, is a function of the digital content and a secret value shared between the sender and the receiver. The MAC generates an *authentication token* that is sent with the digital content [1][2]. The token permits the receiver to verify that the sender sent the digital content and that the digital content has not been modified. Different digital content that generates the same token is called a collision. It should be computationally infeasible to generate a collision. Therefore, the size of the token should be large enough to prevent an adversary from using a general attack such as the Birthday attack to create a collision [1].

The composite message digest and corresponding MAC described in this paper is a basic component of a complete system for implementing a paperless environment in which all digital content is authenticated [3]. The message digest is used to create a MAC, but the focus is on the message digest ensuring the integrity of the digital content. The authentication aspect of the MAC will be discussed in a future paper. The composite message digest is constructed by concatenating multiple existing algorithms so that the probability that two different digital contents have the same message digest is low. Two common algorithms not typically used for generating message digests, CRC and Adler, are tested. A composite message digest is successfully constructed using CRC algorithms. It is difficult for an adversary to create a collision when using this message digest.

Van Oorschot and Wiener presented a method for performing a parallel collision attack in [4]. In 1994, they proposed an algorithm and a custom machine that cost \$10 million that could break MD5 in 24 days. Today this machine would cost \$14 million. Table 1 presents the mean number of steps that would be necessary to find a collision and the approximate amount of time to find one for different size message digests using the method in [4] for years 2003 and 2024. As in [4], it is assumed that 350,000 special purpose processors could be built. The number of steps performed each second by each of these processors was changed to values that represent technology in 2003 by using Moore's Law that states that processor speed doubles every 18 months. Therefore, the processor speed was set to 3x10^9 steps per second for year 2003 and scaled accordingly for year 2024. It is assumed that the complexity of the algorithm that can find a collision is

$$O\left(\left(\sqrt{\pi \times 2^k/2}\right)/p\right), \tag{1}$$

where k is the number of bits in the token and p is the number of processors [4].

Table 1 Mean Time to find a Collision

# bits	E[steps]	Year 2003	Year 2024
128	6.6x10^{13}	6 hours	1 second
160	4.3x10^{18}	46 hours	1 day
192	2.8x10^{23}	3.0x10^6 years	180 years
256	1.2x10^{33}	1.3x10^{16} years	7.9x10^{11} years
320	5.2x10^{42}	5.5x10^{25} years	3.4x10^{21} years

Table 1 demonstrates that an off-line special-purpose machine could find a collision for 128-bit hash function like MD5 in six hours in 2003 and in one second in 2024. A collision for 160-bit hash function like SHA-1 could be found in one day by year 2024. A goal in this paper is to construct a message digest and corresponding MAC that can sustain such an attack over a 20-year period.

2. BACKGROUND

There are two methods of providing authentication [1]. First, digital content authentication can be provided by using a hash function in combination with encryption. A hash function maps a string of arbitrary length into a fixed-length string [5]. The output of a hash function is called a *hash value*, *checksum*, *fingerprint*, or *message digest*. The message digest is then encrypted by either using symmetric or asymmetric encryption to create a token. The digital content and encrypted message digest are then sent to the receiver. The receiver decrypts the message digest and compares it with the message digest that it independently calculates from the digital content to verify that the digital content is authentic.

The second method, which is the focus of this paper, is to use a Message Authentication Code (MAC), which is a function of the digital content to be authenticated and a secret shared value shared by the two communicating parties to create an authentication token [1]. By incorporating a secret value in the MAC, an adversary is prevented from modifying the digital content without detection. There are several reasons to use a MAC instead of the encryption method for authentication [5]. First, encryption software is slow. Authentication can be implemented with or without secrecy with a MAC. A second reason to use a MAC is that patents cover many encryption algorithms. Finally, encryption algorithms are subject to U.S. export control, which hinders their use for international business.

In this paper, existing functions and data transformations are used to construct a larger message digest and corresponding MAC. The cyclic redundancy check (CRC) [6] and Adler [7] functions are used. Both algorithms require an initial vector (IV), so the IV is the shared secret value. Also, compression is used to transform the original digital content into another string and the functions are applied to the transformed digital content. Optimization is disabled in the compression routines such that the size of the digital content that cannot be compressed will increase. The individual tokens of the functions are concatenated together to create the larger token.

Not counting the length qualifiers, the length of the complete authentication token consisted of six 32-bit values for a total of 192 bits. The big MAC with a static IV is applied to 64 million randomly generated files to measure the number of collisions. No collision was found for all 192 bits, but collisions were found for the individual 32-bit functions. The number of collisions for the individual functions is compared with the theoretical number of collisions as predicted by the Birthday paradox. The CRC function performs as predicted, but the Adler function performs poorly as a hash function generating approximately seven times as many collisions as predicted by the Birthday paradox. Therefore, the Adler function was first replaced with a CRC function with the same generator polynomial and a different initial vector. A successful attack easily created a collision. Then the Adler function was replaced with a CRC function that used a different generator polynomial than the other CRC function and it performed as predicted by the theory.

2.1. Hash functions

Two common hash functions are MD5 and SHA-1. The MD5 hash function takes an arbitrary length of digital content and produces a 128-bit message digest [8]. The secure hash algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and is published in FIPS 180-2 in August 2002. The standard describes SHA-1, SHA-256, SHA-384, and SHA-512, which have message digests of 160, 256, 384, and 512 bits, respectively [9].

2.2. Message authentication codes

2.2.1 Universal hashing: Universal hashing can be used to generate a MAC. In universal hashing, instead of one hash function being used, a hash function is chosen randomly from a set of functions that have specific properties [10]. If the hash function is chosen randomly each time, then the average performance of the hashing operation is good for any distribution of input digital content. This helps average out the performance of hashing over several input digital contents with different distributions. Using universal hashing, a MAC can be constructed that is unbreakable with certainty p [10].

There are two methods described in [10]. First, each time digital content is sent, the sender and receiver exchange a secret value that identifies which hash function to use to authenticate this digital content. The sender creates the token using the secret hash function and sends it and the digital content to the receiver. The receiver uses the same hash function to create a token and compares it with the received token to authenticate the digital content.

The second method presented in [10] is to number each digital content with a number b that has the same bit length as the token. Now there are two secret values to be exchanged, the hash function and the digital content number. The first is the particular hash function f being used and the second is the number of the digital content b. The number b must only be used once with a particular function so it is also considered to be a one-time pad. The token t is constructed by applying the hash function f to the digital content $m, f(m)$ and then XORing the message digest with the number b. Therefore, the token is $t = f(m) \oplus b$, where $\oplus$ is the exclusive-or operation. This permits the sender to be able to use the same hash function as long as the number b is changed every time.

2.2.2 HMAC: Another construction to create a MAC is called the Hash-based Message Authentication Code (HMAC) [2]. It can use a hash function to provide a practical and efficient MAC. HMAC uses a hash function such as MD5 or SHA-1 that does not have an inherent key or secret value to create a MAC by setting the function's initial vector (IV) to a secret value. HMAC is popular and has gained wide acceptance. It is used to implement authentication for Internet security protocols [9]. Part of the reason that HMAC is successful is that many countries restrict the import and export of software that use keyed hash functions [11]. They do not restrict keyless hash functions. Therefore, HMAC uses a keyless hash function to build a keyed hash function. NIST sanctions the use of HMAC when using the SHA-X series for authentication [12].

2.2.3. MAC based on CRC: Krawczyk presented two simple and efficient hash functions that can be used for a MAC [13]. One is based on Toeplitz hashing and the other is a cryptographic version of the cyclic redundancy code (CRC). Either method can be applied in two different ways. First, a hash function is chosen secretly from a family of universal hash functions for each separate digital content. Alternatively, the same hash function is used to send multiple digital contents, but the resulting message digest is encrypted with a one-time pad to prevent an adversary from discovering which hash function was chosen. The second method usually requires generation of less random bits and the hash functions can be generated off-line. For practical reasons, the one-time pad can be implemented by secretly exchanging a seed to a pseudorandom number generator.

The cryptographic CRC or MAC presented in [13] is constructed by treating digital content m as a series of bits $m(x)$. The secret value is an irreducible polynomial $p(x)$ of degree k over Galois field, GF(2). Then the MAC using a CRC is defined as Eq. (2).

$$CRCMAC(p(x), m(x)) = m(x) \bullet x^k \bmod p(x) \quad (2)$$

Note that multiplying the digital content by x^k using modular arithmetic causes the binary representation of the digital content to be shifted k values to the left. *This operation must be performed or the CRC can be easily broken* [13]. Without performing the shift any of the lower k bits of the message can be modified and the corresponding position bit in the token can be modified. Note that this can be performed after encrypting the message digest with the one-time pad to successfully forge the digital content. This linear property was also used in [14] to modify packets in a wireless network so that the CRC would not detect tampering.

The irreducible polynomial p(x) must be kept secret. As shown in [15], the CRC process can be worked backwards to find a consecutive sequence of bytes that will make the CRC message digest of different digital content be the same as the CRC message digest of the original digital content with the same length.

2.3. Birthday attack

In this paper, it is assumed that the MAC maps the digital content and the secret value into a fixed number of k bits [1]. A general attack commonly called the Birthday attack can be used against a hash function or a MAC [2]. It is based on the Birthday paradox from probability theory [1]. An adversary could generate N valid digital contents and N bad digital contents with the corresponding message digests or MACs depending upon whether a hash function or MAC is used. The probability that some bad digital content would generate the same message digest or MAC as one of the valid digital contents is approximately equal to 0.5 as N approaches $2^{k/2}$. Therefore, the number of possible values represented by the k bits of the MAC must be sufficiently large so that the computation of $2^{k/2}$ variations of digital content is infeasible.

When using the Birthday attack against a message digest an adversary can work off-line to find collisions because the hash function and the initial vector (IV) are known. This type of off-line attack can become feasible since the attack can be easily performed in parallel [2][4] as discussed in the Introduction.

However, what if the sender or receiver wants to forge the MAC? They both have the secret value, so either could perform an off-line attack to find collisions for an extended amount of time and then claim the other one sent the new digital content. A goal is to construct the composite MAC so that it can withstand an attack over an extended amount of time.

In this paper, 64 million random files are generated and different variants of the composite message digest are applied to look for collisions. Since the message digest is a composite of smaller functions, the number of theoretical collisions is compared with the actual number of collisions for the individual functions. Let m be the number of files. Let k be the number of bits in the token. For m random files, the probability of finding a collision given a particular token is equal to Eq. (3).

$$P[collsion \mid token] = 1 - \left(\frac{2^k - 1}{2^k} \right)^{m-1} \quad (3)$$

This quantity can be approximated by using the inequality in Eq. (4).

$$1 - x \le \exp(-x) \quad (4)$$

which results in

$$P\left[collsion \mid token\right] \approx 1 - \exp\left(-\left(\frac{m-1}{2^k}\right)\right). \qquad (5)$$

Then, the theoretical mean number of collisions $E[collisions]$ for m random files and a token that has k bits is approximately equal to Eq. (6).

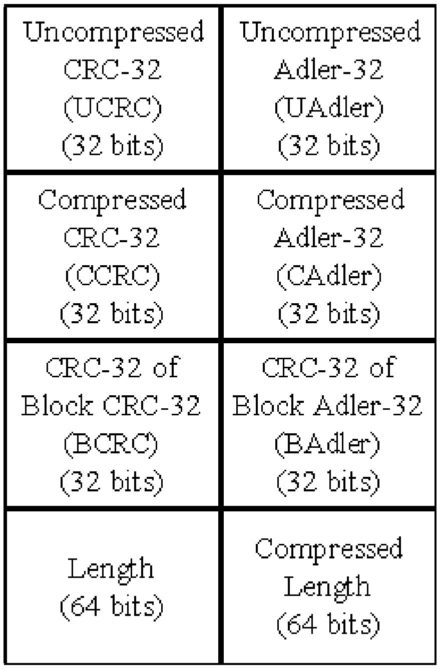
$$E\left[collsion\right] \approx m\left[1 - \exp\left(-\left(\frac{m-1}{2^k}\right)\right)\right] \qquad (6)$$

2.4. Building a big MAC

One way to create a larger MAC that reduces the probability of collision is to apply a MAC to digital content with two or more independent keys [16]. For example, a MAC with a token that has k bits has a probability of collision using the Birthday paradox of $1/2^{k/2}$. If the MAC is applied with two independent keys and the token consists of the two tokens concatenated, then the probability of collision decreases to $1/2^k$ [16]. Another way to reduce the probability of collision is to apply multiple members of the universal hash-function family to the digital content and concatenate the results [17]. In this case, the secret values are which members of the universal hash function that were used.

3. PROPOSED BIG MAC

The proposed big MAC consists of six 32-bit hashes and two 64-bit length qualifiers and is shown in Figure 1. The length qualifiers will be ignored. The first two components of the MAC are named uncompressed CRC-32 and uncompressed Adler-32 because they are applied to the uncompressed digital content. The second two components are named compressed CRC-32 and compressed Adler-32 because they are applied to a compressed version of the digital content. The digital content is compressed using LZSS [18], and then the CRC-32 and Adler-32 are computed. Optimization is eliminated in the compression routines so the file size grows for digital content that cannot be compressed. The next two components are a combination of two hash functions. The first is a CRC-32 of CRC-32 blocks. The CRC-32 is used on blocks of 4 bytes to create multiple tokens, which are concatenated and then the second CRC-32 hash function is applied to these tokens. The next component is a CRC-32 of Adler-32 blocks, where the blocks are 4 bytes. A CRC-32 hash function is applied to the concatenated tokens calculated with Adler-32. The last two components are the length and compressed length qualifiers, where the length is the digital content length in bytes and the compressed length is the length of the digital content compressed with LZSS in bytes.



Uncompressed CRC-32 (UCRC) (32 bits)	Uncompressed Adler-32 (UAdler) (32 bits)
Compressed CRC-32 (CCRC) (32 bits)	Compressed Adler-32 (CAdler) (32 bits)
CRC-32 of Block CRC-32 (BCRC) (32 bits)	CRC-32 of Block Adler-32 (BAdler) (32 bits)
Length (64 bits)	Compressed Length (64 bits)

Figure 1 Proposed Big MAC

4. TESTS

4.1. Simple MAC forgery when using CRC

The method used in [13] to forge a MAC was verified on several files. Several files were changed by one bit in the least significant four bytes because the proposed MAC is composed of 32-bit CRC and Adler functions. Then the token was changed in the same position as the modified file to successfully forge the CRC component of the MAC.

Krawczyk proposes a simple fix to prevent the simple forgery that is described above [13]. Appending k zeros to the end of the message, where k is the length in bits of the CRC message digest, will randomize the bit needed to change so that is it harder to forge. The same test files used to verify the method of changing one bit to forge a file were used to test the method described in [13] of adding k zeros. Four bytes of zeros were added to the files and the MAC was applied again. As expected, the CRC function could not be easily forged.

4.2. Distribution of collisions

Next, three tests were performed to see what affect adding the zeros would have on the distributions of the component tokens. First, 64 million random 1-KB files were created and their corresponding tokens. Next, the same files were used, except the last 4 bytes were changed to zeros. Then, 4 bytes of zeros were appended to the 1-KB files and the MAC was applied. The results are shown in Figure 2.

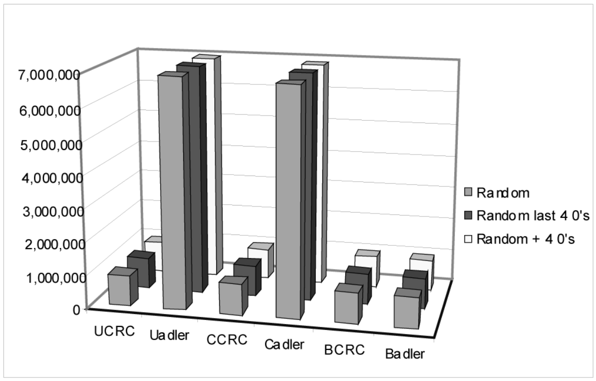


Figure 2 Distributions of Collisions on the big MAC

The mean number of collisions for a 32-bit hash is 946,604 for 64 million random files. Since this big MAC is composed of multiple 32-bit functions, collisions do occur for the individual functions. Notice that the uncompressed Adler-32 and compressed Adler-32 have about seven times as many collisions as the other components of the MAC.

4.3. Adler replaced with CRC

Next, the Adler-32 hash was replaced with CRC-32 since the Adler-32 hash function had about seven times as many collisions as predicted by the Birthday paradox. First, the 64-million-message test was performed using the same polynomial for both of the CRCs, but different initial vectors. In this test, the polynomial 0x04C11DB7 is used for both CRC-32 hash functions. The initial vector for the first CRC-32 was set to 0xFFFFFFFF and the initial vector for the second CRC was set to 0x55555555.

The next test used different polynomials for the two CRC-32 functions, but the same initial vector. The polynomial for the first CRC-32 was set to 0x04C11DB7 and the polynomial for the second CRC-32 was set to 0xA288C559. The seed 0xFFFFFFFF was used for both CRC-32 hash functions. The results are shown in Figure 3.

The number of collisions for the individual functions is approximately the same for both of the tests. Also, the empirical data is approximately equal to the predicted number of collisions of 946,604. Thus, using a CRC function instead of the Adler-32 hash function provides a MAC with a more uniform distribution. As seen in Figure 3, the tokens on the compressed messages, CCRC and CCRC2, have more collisions than the tokens on the uncompressed messages because of the smaller data set. Notice that the blocked hash functions, BCRC and BCRC2, have slightly less collisions. This is because the input stream from the first application of the CRC causes the input to the second hash function to be more uniform.

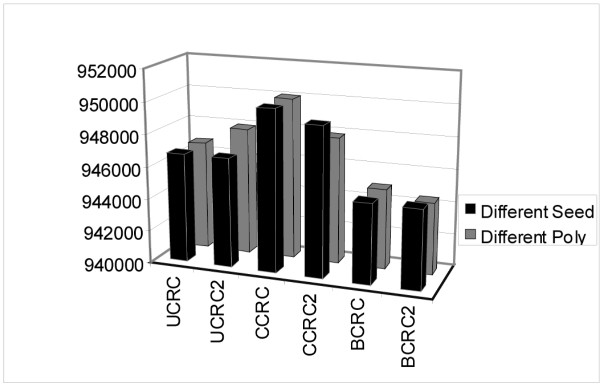


Figure 3 Distributions of collisions replacing Adler with CRC

4.4. CRC attack on same CRC polynomials with different seeds

The CRC attack described in [15] was performed on the files described in the previous section to change a byte and add four additional bytes to maintain the same token and message length. The attack was performed on the MAC that used the same CRC polynomial 0x04C11DB7 with different seeds.

Both polynomials were set to 0x04C11DB7 and the initial vectors were set to 0xFFFFFFFF and 0x55555555, respectively. There were a total of 64 million tokens generated. No full collisions were found. However, both the CRC that used an initial vector of 0xFFFFFFFF *and* the CRC that used an initial vector of 0x55555555 were compromised. In other words, a byte was changed in the file and four additional bytes were placed in the file that caused two functions of the composite MAC to be the same as the original message. *So the same CRC polynomial with two different IVs does not provide a good big MAC.*

The reason that the attack on one of the CRC functions simultaneously works on the other CRC function can be explained by the cyclic nature of the CRC. With the same generator polynomial, a CRC generates a cycle of bit patterns depending upon the input file. The initial vector just picks a position in the cycle. Therefore, the CRC attack in [15] reverses both CRC sequences causing two different files to have the same pair of CRC tokens.

4.5. CRC attack using different CRC polynomials with the same initial vector

In the second test, the polynomial 0x04C11DB7 was used for CRC1 and Adler was replaced with the CRC2 hash function using the polynomial 0xA288C559. The initial vector 0xFFFFFFFF was used for both CRCs and 64 million tokens were generated. CRC1 was compromised by the CRC attack, but CRC2 was unaffected by the attack. The attack did not work because a different generator polynomial was used. Therefore, it is suggested that when building a composite MAC with CRC hash functions that different polynomials should be used.

5. CONCLUSIONS

In this paper, a larger message digest was constructed by concatenating the output of multiple CRC functions using different generator polynomials to decrease the probability of collision using a method similar to [13]. It also demonstrated that adding the same number of zeros as the CRC message digest prevents a simple forgery. Even though the larger composite message digest uses CRC functions, which are not typically used because of their lack of cryptographic strength, it performs well against a brute-force attack. Different functions, such as MD5 or SHA-1, could be used instead to create a larger message digest and corresponding MAC, if necessary. Finally, the technique of concatenating multiple functions to build the larger message digest can be used to construct a variable-size message digest depending upon the required strength.

6. ACKNOWLEDGMENTS

The authors would like to thank Logical Dynamics, Inc., for the opportunity to collaborate on the authentication research. The material is based upon work supported by the National Science Foundation under Grant No. 0090596.

7. REFERENCES

[1] R. R. Jueneman, S. M. Matyas, and C. H. Meyer, “Message Authentication,” *IEEE Communications Magazine*, vol. 23, no. 9, Sep. 1985, pp. 29-40.

[2] M. Bellare, R. Canetti, and H. Krawczyk, “Keying hash functions for message authentication,” in *Proceedings of CRYPTO ’96*, Lecture notes in Computer Science, vol. 1109, N. Koblitz ed., Aug. 1996, New York: Springer-Verlag.

[3] Logical Dynamics, Inc., http://www.logidyn.com

[4] P. Van Oorschot and M. Wiener, “Parallel collision search with applications to hash functions and discrete logarithms,” in *Proceedings of the 2nd ACM Conf. Computer and Communications Security*, Nov. 1994, Fairfax, VA.

[5] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 3rd ed., Prentice Hall, 2003.

[6] S. Lin, D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 1982.

[7] J. G. Fletcher, “An arithmetic checksum for serial transmissions,” *IEEE transactions on Communications*, vol. COM-30, no. 1, Jan. 1982, pp. 247-252.

[8] R. Rivest, *The MD5 Message-Digest Algorithm*, RFC 1321, Apr. 1992.

[9] FIPS 180-2, *Secure Hash Standard*, NIST, Aug. 1, 2002.

[10] M. N. Wegman and J. L. Carter, “New hash functions and their use in authentication and set equality,” *Journal of Computer and System Sciences*, vol. 22, 1981, pp. 265-279.

[11] M. Bishop, *Computer Security: Art and Science*, Pearson Education, 2003.

[12] FIPS 198, *The Keyed-Hash Message Authentication Code (HMAC)*, NIST, March 6, 2002.

[13] H. Krawczyk, “LFSR-based hashing and authentication,” in *Proceedings of CRYPTO ’94*, Lecture notes in Computer Science, vol. 839, Aug. 1994, New York: Springer-Verlag, pp. 129-139.

[14] N. Borisov, I. Goldberg, and D. Wagner, “Intercepting mobile communications: the insecurity of 802.11,” in *Proceedings of the seventh annual international conference on Mobile computing and networking*, 2001, Rome, Italy, pp. 180-189.

[15] B. Maxwell, D. R. Thompson, G. Amerson, and L. Johnson, “Analysis of CRC methods and potential data integrity exploits,” in *Proceedings of the International Conference on Emerging Technologies*, Aug. 25-26, 2003, Minneapolis, MN. http://www.rfbinternational.com/papers/thompson.pdf

[16] S. Halevi and H. Krawczyk, “MMH: software message authentication in the Gbit/second rates,” in *Proceedings of the 4th Workshop on Fast Software Encryption*, Lecture notes in Computer Science, vol. 1267, 1997, New York: Springer-Verlag, pp. 172-189.

[17] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, “UMAC: Fast and secure message authentication,” in *Proceedings of CRYPTO ’99*, Lecture notes in Computer Science, vol. 1666, Aug. 1999, New York: Springer-Verlag.

[18] Storer, J. A. and Szymanski, T. G., “The macro model for data compression,” in *Proceedings of 10th ACM Symp. on Theory of Computing*, 1978, San Diego, CA, pp. 30-39.