A HIGH SPEED MANIPULATION DETECTION CODE

Robert R. Jueneman
Computer Sciences Corp.
3160 Fairview Park Drive
Falls Church, VA 22042
(703) 876-1076

Abstract

Manipulation Detection Codes (MDC) are defined as a class of checksum algorithms which can detect both accidental and malicious modifications of an electronic message or document. Although the MDC result must be protected by encryption to prevent an attacker from succeeding in substituting his own Manipulation Detection Code (MDC) along with the modified text, MDC algorithms do not require the use of secret information such as a cryptographic key. Such techniques are therefore highly useful in allowing encryption and message authentication to be implemented in different protocol layers in a communication system without key management difficulties, as well as in implementing digital signature schemes. It is shown that cryptographic checksums that are intended to detect fraudulant messages should be on the order of 128 bits in length, and the ANSI X9.9-1986 Message Authentication Standard is criticized on that basis. A revised 128-bit MDC algorithm is presented which overcomes the so-called Triple Birthday Attack introduced by Coppersmith. A fast, efficient implementation is discussed which makes use of the Intel 8087/80287 Numeric Data Processor coprocessor chip for the IBM PC/XT/AT and similar microcomputers.

Key words: Manipulation Detection Code (MDC), Message Authentication Code (MAC), checksums, birthday problem attacks, authentication, encryption, digital signature, cryptography, numeric data processor chip, math coprocessor chip, 8087, 80287, IBM PC.

A.M. Odlyzko (Ed.): Advances in Cryptology - CRYPTO '86, LNCS 263, pp. 327-346, 1987.

1 Introduction

A common theme throughout a series of papers[1,2,3] by the author and his colleagues, Dr. S. M. Matyas and Dr. C. H. Meyer of IBM, has been the desirability of separating the function of encryption from that of authentication, so that they could operate at different architectural layers or levels in an communications system. In the context of the ISO Open System Interconnect reference model, for example, it was suggested that link encryption might be applied to all of the communications from a host, using a stand-alone link encryption device operating at ISO OSI layer 1, the data link layer. In this case the appropriate place for authentication would probably be in the Presentation or Application layers (layer 6 or 7), implemented in an application program inside the host. We have also suggested that since the mode of encryption might change depending on the physical medium involved, it would be desirable if the method of authentication were independent of the encryption scheme used.

The recently announced decision of the National Security Agency not to endorse new DES equipment for certification in accordance with Federal Standard 1027 after 1988, and in general to move on to a new family of encryption algorithms for both Unclassified, National-Security Related traffic as well as classified data, should serve to underscore the advisability of such a separation of function, as it will result in an increased requirement for "keyless" Manipulation Detection Code algorithms. Until the new Commercial COMSEC Endorsement Program (CCEP) algorithms are widely available (and perhaps for an even longer period, in the case of international circuits which may have to continue running DES), application programs might be supported by two or even three different link encryption algorithms (DES, an unclassified CCEP Type 2 algorithm, and a classified CCEP Type 1 algorithm, depending on the destination), but should require only one authentication algorithm. It should be observed that there is a fundamental difference between encryption and authentication with respect to the need to change algorithms, for in the case of encryption it is very difficult to know whether your traffic is being broken surreptitiously. In the case of authentication, however, it usually becomes obvious sooner or later if you have been spoofed. The objective is to minimize the amount of time required to detect the spoofing. It would therefore seem that authentication algorithms would not have to be changed nearly as often as encryption algorithms, and that there is perhaps less need for secrecy in their design.

In the papers presented to date, our primary concern was to find an authentication algorithm that would be more efficient than a MAC (especially when implemented in software on a microprocessor), and/or would not require a traditional encryption operation. Only secondarily did we focus on what this author now believes to be the fundamental distinction between an MDC and a MAC, i.e., that whereas a MAC involves one or more secret keys, *an MDC makes use*

[1] Jueneman, Robert R., "Analysis of Certain Aspects of Output Feedback Mode", **Advances in Cryptology: Proceedings of Crypto82**, Plenum Press, New York, 1983, pp 99-127.

[2] Jueneman, R. R., C. H. Meyer, and S. M. Matyas, "Message Authentication With Manipulation Detection Codes", **Proceedings of the 1983 IEEE Symposium on Security and Privacy**, IEEE Computer Society Press, 1984, pp 33-54.

[3] Jueneman, R. R., C. H. Meyer, and S. M. Matyas, "Message Authentication", **IEEE Communications Magazine**, Sept. 1985 - Vol. 23, No. 9, pp 29-40.

of only publicly known quantities, and is therefore considerably more convenient from the standpoint of key management.

1.1 Cryptographic Checksum Requirements

Let us assume that we wish to apply a cryptographic seal to some electronic message or document, and that we will either use a digital signature approach, or else use link or end-to-end encryption to protect the MDC result. We must assure that the set of all checksums is very nearly one to one with respect to the set of all message texts, so that we can easily check the checksum (for example in the digital signature) instead of having to process the entire text. That is, given two messages A and B with checksums, we desire that checksum (A) and checksum (B) be identical if and only if the messages A and B are themselves identical. Assuming a good checksum algorithm, the chances that A and B are *not* identical given that checksum (A) equals checksum (B) should be 2^{-k}, where k is the number of bits in the checksum and the probabilities are averaged over all possible messages.

More specifically, the algorithm should have the following properties:

1. If two different texts (of arbitrary length) are checksummed, the probability that the two checksums will be the same when the two documents are not identical should be a uniformly distributed random variable that is independent of the text, with an average value over all possible texts of 2^{-N} where N is the number of bits in the checksum.

2. The checksum must be sensitive to permutations, so that the message ABC will produce a different value than ACB, etc.

3. As will be seen, the resulting checksum must be on the order of 128 bits in length, in order to resist a so-called "birthday attack" against the text itself.

4. Finally, all of the bits of the checksum must be an over-determined function of all of the bits of the text and all of the bits of the checksum of the previous block, in order to defeat several attacks that will be discussed below.

In addition, in a number of applications it is necessary to add a random Initialization Vector to the text itself, and to chain the blocks of messages together by including the checksum of the previous block in the checksum of the current block, so that one properly authenticated value cannot be substituted for another in a *playback attack*. For example, if a particular dialog occurs frequently, and the answer to some question is either "Yes" or "No", without the appropriate chaining the attacker could easily substitute the entire contents of a previous message, together with its valid checksum, and the message would be accepted. A 64-bit random Initialization Vector will suffice to initialize the authentication, but message chaining may still be required. It should be noted that an Initialization Vector may also be necessary to ensure that the same text is encrypted differently each time it is transmitted, in order to prevent a so-called dictionary attack. In general it appears that the same Initialization Vector (sometimes called a Message Indicator) could be used for both purposes, but it would be necessary to carefully examine both the encryption and the authentication scheme before making a blanket statement.

Finally, we must point out that although a DES-based Message Authentication Code or MAC could be used to authenticate either an encrypted or unencrypted text without further encryption because it makes use of a secret key[4], that is not true of a Manipulation Detection Code. Although the text itself does not need to be encrypted, the MDC must be, so that the attacker cannot substitute his own MDC with any significant probability of success. In most cases, the MDC can simply be appended to the message, and if the entire message is encrypted together with the MDC, that will provide adequate protection. If the MDC is easier to calculate than an MAC, then if the message would be encrypted for secrecy in any case the MDC technique would be more efficient than a MAC.

2 Attacks Against Checksum Techniques

In the three previous papers in this series, we have addressed different aspects of the problem of authenticating the contents of a message against possible modification or corruption. In the first, a flaw in a draft of a federal standard regarding Manipulation Detection Codes was pointed out briefly, and a quadratic residue technique suggested as an alternative form of checksum. That paper also pointed out the need for two independent keys for encryption and authentication if a Message Authentication Code (MAC)[5] is generated through the use of a secret (DES) key and appended to the message, for it was shown that the errors introduced in the plaintext by an error or by manipulation were exactly the errors needed to cause the MAC to be erroneously computed so as to validate the manipulated text.

The second paper presented an extensive analysis of various forms of Manipulation Detection Codes, including block XOR and linear addition techniques, when used in combination with Cipher Block Chaining, Cipher Feedback, and Output Feedback modes. That paper also discussed the architectural advantages of a Manipulation Detection Code that was independent of an encryption algorithm, particularly in those cases where low-level link encryption may be used to protect the traffic flowing into or out of a main-frame host processor, yet it is desired for an application program in the host to verify the authenticity of the messages received. In addition, the potential speed advantages of an MDC technique compared to the calculation of a MAC were discussed.

During the course of writing that paper and reviewing it with our peers, a number of attack scenarios were identified that must be considered whenever new schemes are proposed. In particular, Dr. Don Coppersmith introduced several attacks which he called under-determined knapsack attacks. These have also been called "birthday" attacks, because they generally involve generating random variations in the text and calculating a MAC or an MDC, then working

[4] This is not recommended, however, because an *unencrypted* MAC *reveals* something about the message itself, and may form the basis for a dictionary attack.

[5] As defined in Federal Information Processing Standard FIPS PUB 46, *"DES Modes of Operation"* published by the National Bureau of Standards, "A MAC may be generated using either the CFB [Cipher Feedback] or CBC [Cipher Block Chaining] mode. In CFB authentication, a message is encrypted in the normal CFB manner except that the cipher text is discarded. After encrypting the final K bits of data and feeding the resulting cipher text back into the DES input block, the device is operated one more time and the most significant M bits of the resulting DES output block are used as the MAC, where M is the number of bits in the MAC. In CBC *authentication*, a *message* is encrypted in the normal CBC manner but the cipher text is discarded. Messages which terminate in partial data blocks must be padded on the right (LSB) with zeros. In CBC authentication, the most significant M bits of the final output block are used as the MAC."

forward and backward until two matching MACs or MDCs are found. Making random variations in the text in two places and then sorting and comparing the results for a match allows the attacker to take advantage of the so-called Birthday Problem in statistics to reduce the work required to approximately the square root of the effort required to match a particular given MAC or MDC.

2.1 The Fundamental Birthday Attack.

The third paper abstracted the second for a more general audience, but also added some new information. In particular, it was recognized that *any* Manipulation Detection Code (MDC) or Message Authentication Code (MAC) is susceptible to a birthday attack against the text itself, unless the MDC or MAC is on the order of 128 bits in length. This fundamental attack proceeds as follows, and assumes that one user is attempting to defraud another by devising a version of a bogus or unfavorable contract or agreement which would have an identical checksum as would an acceptable version of a legitimate one, having the other party digitally "sign" the legitimate version, and then produce the bogus version in front of a judge and claim that the other party has defaulted on his obligations:

1. Assume that a 64-bit MAC or MDC is used, and that if necessary the attacker can exercise the authentication system *ad infinitum* to generate a MAC or an MDC, even if a secret key which he does not know is used in the case of the MAC.

2. The attacker secretly prepares a number of subtle variations of the legitimate text in advance, and calculates (or has the system calculate) the MDC or MAC for each one. In the case of an electronic mail message or document, for example, suppose that a number of lines contain the ASCII character sequence "space-space-backspace"[6] between selected words. The attacker might prepare a set of variations of that document in which the sequence in selected lines would be "space-backspace-space". The length of the text would not be altered thereby, and all of the variations of the document would appear to be identical, both when printed and when displayed on the normal video display, unless "dumped" in hexadecimal format. Other, more consequential changes to the text could also be made, of course. By systematically altering or not altering the text in each of say 32 different lines, 2^{32} or 4.3 billion variations could be generated. A file of records consisting of the MAC or MDC plus a 32-bit permutation index could be used to summarize what lines were altered by a given variation, and what MAC or MDC resulted.

3. The attacker then prepares an equally large number of variations on the bogus text he would like to substitute for the legitimate text, and calculates (or has the system calculate) the MDC or MAC for each one of those variations as well, producing another file of MAC/MDC results plus the permutation index records.

4. The attacker then compares the two files, searching for a pair of identical MACs or MDCs and noting the permutation indices. (If no match is found, the attacker can simply generate a few more random variations of the legitimate and the bogus texts until a match

[6] Other combinations, such as null-character, or carriage return - line feed would also work, as well as less subtle variations such as changing "the" to "an", or inserting or deleting commas or spaces in a numeric field.

is found.) He then recreates the full text of both the acceptable and the unacceptable documents with the specific modifications necessary to produce the matching MACs or MDCs, based on the permutation indices.

5. Finally, he offers the appropriate variation of the legitimate contract to the other party and both "sign" it. At some time in the future the attacker substitutes the unfavorable contract, and tells the judge that the digital signature containing the MAC/MDC "proves" it was that version that was signed by both parties.

This is Yuval's[7] classic "How to Swindle Rabin" form of a so-called "Birthday Problem" attack.

According to the famous birthday paradox[8] problem in statistics, this kind of an attack is likely to succeed if the number of variations of each document that are generated and compared approaches the square root of the total number of possible MAC/MDC values. That is, if a 32-bit checksum were used, the probability of a successful attack would be about 50% after only 2^{16} or 65536 variations were computed, and would increase rapidly after that point. If a 64-bit MAC or MDC were used, then the 4.3 billion iterations produced by systematically varying 32 lines of text would be likely to suffice.

In order to see whether this attack would be computationally feasible against a 64-bit MAC, let us assume that the variations all occur at the end of the text and that exactly one variation occurs in 8 bytes of text, so that only one DES iteration would be required to account for that variation. The brute-force way to calculate the resulting MAC for the entire text would be to recalculate the last 32 DES blocks for each variation, which would require 2 x 32 x 2^{32} DES iterations for the two sets of variations of the text. However, by only encrypting those blocks that have changed and those for which earlier blocks have changed, the number of DES iterations can be reduced to 2 x (2^{33}-1). A hardware DES implementation running at 10 microseconds per iteration could complete the task in just under 2 CPU days.

However, the amount of I/O required to sort and compare the data must not be neglected. A 64 bit MAC and a 32 bit permutation index per variation would require 12 bytes per entry times 2^{32} entries, or 51.5 gigabytes per file. At an effective rate of 20 microseconds per variation (including encrypting due to the requirement to reencrypt blocks after a change), data would be generated at the rate of 4.8 Mbps or 600 kilobytes per second, which is well within the channel capacity of a mainframe computer to record. The process of comparing two files consisting of 340 reels each of 6250 bpi high-density tape (151 megabytes per reel), searching for any one value on one file that matches any one value on the other file, would admittedly be a lengthy task even for a mainframe computer, but it is not infeasible. One approach would be to presort the information by distributing the data across 22 tape drives while the information is being generated, producing 22 files of approximately 15 to 16 reels each for each variation. Each of those files could in turn be distributed onto 20 reels of tape at maximum tape speed, and then those approximately 680 individual reels could be sorted one at a time using a conventional tape

[7] Yuval, G., "How to Swindle Rabin", Cryptologia, Vol 3., No. 3, July 1979, pp 187-190.

[8] How many people must there be in a room in order to have a good chance that at least two people in the room will have the same birthday.

or disk sort routine, and finally compared. Assuming each reel requires 15 minutes to sort, the total process could be completed in about a week.

An interesting alternative technique was suggested by Caron and Silverman's distributed processing approach to factoring[9]. Let us assume that the attacker has at least the occasional use of 256 Intel 80386-based microprocessors or similar machines which are connected via a high-speed LAN. Each of these slave machines will be assumed to have two boards of 8 megabytes[10] each of the new 1 megabit memory chips. In addition, a master station will be equipped with a hardware DES implementation, four 8-megabyte memory boards, and two 85 megabyte hard disks.

The total amount of memory in the 256 slave processors would be 4.295 gigabytes, or 2^{35} bits. Let us assume that after each calculation of a MAC in the first set of variations, the master workstation sends 24 bits (bits 8 through 32) of the MAC to the appropriate slave processor based on bits 0 to 7 of the MAC. Each slave processor would then use those 24 bits to address a particular bit within its memory, and would turn on that bit. At the end of the first pass through all of the variations of a single document (requiring about 24 hours), the contents of the first 32 bits of all 2^{32} MACs calculated would be represented as a set of bits turned on in all of the memories. Because there are 2^{32} bits turned on out of 2^{35} bits total, the probability that a particular bit will be on after the first pass is 1/8, with many bits having been turned on multiple times within this pass. At the end of the first pass, all of the slave processors would dump memory to a hard disk, then zero all of the bit storage area.

The master processor would then begin processing the second set of variations and would again send 24 bits of the MAC to all of the slave processors. *This time, however, the slave processors would check to see if that particular bit had already been turned on.* If it had, it would signal the master CPU, which would record that permutation index. Because the probability of a particular bit being turn on in both the first and the second passes is 1/64, a 1 byte increment from the previous permutation index would normally suffice and there would be approximately $2^{32}/64$ or 67,108,859 values to record, so one 85 megabyte hard disk would be sufficient to contain one set of permutation indices.

The master CPU would then repeat the calculations of the first pass in a third pass, again broadcasting 24 bits of the MAC to the appropriate slave stations, which would replay whenever a collision was found. The master station would then record the permutation indices associated with those collisions on the second 85 megabyte hard disk.

This entire three pass process would then be repeated, but instead of examining the first 32 bits of the MAC the last 32 bits would be used. The fourth pass would initially turn a set of bits based on the first document, and the fifth pass would check for a possible collision. However, the master CPU would not have to generate all 2^{32} variations, but would only process the variations that were previously recorded as potential matches after the second and third passes. Therefore, instead of taking two days for this processing, it would only take about 45 minutes.

During the fifth and sixth passes, the various slave processors would send back acknowledgements as before, and the master station would erase any permutation index that did not produce a collision. This time, the probability of a false alarm collision is only 1/4096, so the expected number of collisions remaining to be processed is 1,048,576.

The master station would then make two internal passes over the remaining permutation indices for the two different documents, using a hash table lookup scheme to store/search the 64-bit MAC and 32 bit permutation indices.

2.2 Other Opportunities For Birthday Attacks.

Similar attacks could potentially succeed against command and control systems, especially if the attacker is able to send bogus commands and random variables over a channel that cannot be shut down without denying service to the legitimate users as well. An example would be an attacker who attempts to take over or disrupt a communications satellite by sending spurious commands via the Telemetry, Tracking, and Control channel to the satellite in an attempt to get

[9] Caron, Thomas R. and Robert Silverman, "Parallel Implementation of the Quadratic Sieve", **Advances in Computer Science - CRYPTO '86 Proceedings**, Springer-Verlag, Berlin, 1987.

[10] Sixty-four microprocessors with 64 megabytes of memory would be significantly cheaper, but that would be a very specialized system, as opposed to a configuration that might be used for other purposes and could be "borrowed" for our purposes.

it to move out of position, use up all of the maneuvering fuel, go into a spin, etc. There is no easy way that the attacker can be located, and if he is operating out of a foreign country there may be nothing that can be done to stop his transmissions. The attacker can simply send random data, and even if the command link were encrypted there is a possibility that the decrypted information might be accepted as a valid command. Unless a sufficiently long checksum is used, random data and a random MDC or MAC will eventually result in a random command being accepted[11].

Another instance could arise in a multilevel-secure system, where a cryptographic "seal" is applied to an "object", in order to prevent classified information from being disclosed or modified without proper authorization. For example, if the security classification associated with the object could be manipulated by a Trojan Horse program, a classified object's label could be changed to "unclassified", and the information released. Similarly, the contents of a properly marked, unclassified object could be changed and classified information inserted. Because the sensitivity label must be very closely associated with the contents of the object (to prevent a simple cut-and-paste attack), the security seal of the object typically includes both the sensitivity label and the contents of the object as well. In this case, the Trojan Horse program could conceivably manipulate the label together with some innocuous portion of the data, and repeatedly present the information to the cryptographic seal mechanism until two versions, one good and one bad, happened to produce the same cryptographic checksum. The substitution would then be prepared.

2.3 Recommended Length For Cryptographic Checksums.

Based on these attacks, we conclude that it is essential that any MAC or MDC checksum be on the order of 128 bits in length, in order to protect against situations where the opponent could systematically change both the text and the MAC/MDC until he finds a combination that works.

A 128-bit checksum is sufficient, because in addition to the sorting and searching problem rapidly becoming insurmountable (after about 80 bits), the 2^{65} basic MAC/MDC calculations required by the birthday problem attack would not be computationally feasible, even if they were to take only 1 nanosecond apiece. It must be stressed that this attack has nothing to do with the cryptographic strength of the MAC or MDC algorithm, or whether conventional keys, public keys, or no keys at all are used, but only whether the length of the result is sufficient to withstand any computationally feasible number of random "birthday attack" trials.

In this connection, it is worth observing that the recently revised ANSI X9.9-1986 authentication standard[12] specifies the use of a 32-bit MAC, although the future use of a 48-bit or 64-bit MAC is also discussed. In analyzing the protection afforded by that standard, we should consider both external attacks and internal fraud. With respect to an external threat in this environment, a 32-bit MAC is arguably sufficient. Even though an attack against such a system would be likely to

[11] Actually, satellite command processors typically echo the command received back to the ground, and then require an "Execute" command within a certain period to make the received command take effect. Assuming that the Execute command is also encrypted and authenticated it is much less likely that this particular attack would succeed, but the point is clear.

[12] Financial Institution Message Authentication (Wholesale) X9.9-1986 (Approved August 15, 1986), published by the X9 Secretariat, American Bankers Association, 1120 Connecticut Avenue, Washington, D.C. 20036.

succeed after only 65 thousand attempts, hopefully all of the false MACs should generate some alarm, and the investigative agencies would be called in to stop the perpetrator before he (or she!) was successful.

With respect to a possible internal threat or Trojan Horse program, however, it is obvious that if the security of the system were to rest solely on the authentication provided by the MAC, then a 32-bit MAC is grossly inadequate. It should be apparent from the preceding discussion that even a 64-bit MAC would provide inadequate protection from a member bank or insider who might attempt to defraud another institution, if that were the only mechanism used to protect against such attacks. In the banking environment, of course, there are all sorts of reconciliation processes that would presumably uncover such attempts at fraud sooner or later, but in other environments this might not be the case. *System developers are therefore cautioned not to apply the X9.9-1986 authentication standard outside of the specific wholesale banking environment for which it was developed.*

2.4 The Need For Super-Authentication.

It should be noted that if an MDC technique were used to authenticate a message that is protected by Output Feedback (OFB) mode (or worse yet, not protected at all), the opponent could easily calculate a valid MDC to go with the modified text, and append the new MDC to the text at will, since there is no separate cryptographic key used to protect the authentication information. Even though the attacker doesn't know the key used to encrypt the message, if we assume that he does know the plaintext (perhaps because he generated it) he can determine the keystream output from OFB by XORing it with the plaintext, and can then change the keystream to suit his purposes. This particular attack can be defeated by having the system introduce a secret, varying, random component which the opponent doesn't know (an Initialization Vector) into every message, and including that random value in the MDC calculation. The Initialization Vector is not a key, since it doesn't have to be known in advance by either party. It doesn't even have to be deterministic, and it can be discarded by the receiver after the MDC is checked. However, the random value should be at least 64 bits long, so that the attacker cannot discover its value and then the true value of the MDC and therefore the corresponding bits of the key stream by exhaustively trying all possible values of the initial random component.

With this in mind, let us reconsider the delayed transmission OFB attack that was discussed in the second and third papers. That attack made use of a lengthy message whose plaintext was known to the attacker, so that an extensive amount of keystream would become known. The beginning and end of the message would then be jammed, and an invalid message substituted based on the keystream. The invalid message could even contain a random component, since the attacker would have already recovered the keystream bits for that portion of the output.

In order for this attack to succeed, it is necessary for the attacker to precisely synchronize the plaintext and the ciphertext, know the current message sequence number, intercept the ciphertext and block it, jam the portion of the message containing the secret, random component to make it look like a noise burst on the transmission medium, and then fabricate any desired random value, bogus message, and a corresponding fraudulent MDC, and follow it with a valid HDLC frame check sequence. Finally, the end of the message containing any remaining message text, the old MDC, frame check, and the start of the next message would be replaced with random characters to cause another noise burst to be simulated, which would then be rejected by the standard HDLC error recovery mechanism at the receiver.

It should be clear that this real-time interception and modification technique, although difficult to put into practice, could theoretically be applied to *any* MDC scheme that does not involve the use of a secret key for authentication, if the message text being sent is known to the attacker.

Although this attack was previously considered legitimate, and a potentially serious obstacle to the use of an MDC technique, it can only succeed if the message being attacked is considered in isolation, as if it were the only message being sent. In order to defeat the attack it is only necessary to chain the individual messages together in such a manner that a change in one message will affect the MDC in the *next* message. Therefore, instead of the MDC in a given message pertaining to that message, it should instead pertain to the previous message. The MDC contained in the first message should cover the Call Request/Call Acknowledgement or other session establishment message *sent by the other correspondent*, and containing a secret, random component known to that correspondent, or the system at that end. By MDCing something that the other correspondent already knows, the chain is anchored at the beginning, defeating an attack that would systematically change every message in the sequence.

Each MDC should therefore cover not only the data contents of the previous message, but the previous MDC as well, so that changing a single bit of a message will affect all of the MDC results from then on. The MDC for the previous message then satisfies the requirement for a secret, random component in each message if OFB is used. In order to detect an attempt to delete the final message of a session, a unique end-of-session message should be sent that includes the MDC of the previous message, plus the MDC of the end-of-session message itself. If a digital signature capability is implemented, it would be desirable to sign this final message. If the final MDC is digitally signed, then the initial MDC could be a constant. This would avoid the necessity of having a session established in real time so that the other correspondent can check the original value of the MDC at the time of session startup. This would be particularly useful in store-and-forward message systems, including electronic mail and bulletin board systems, where the receiver is not in direct contact with the originator and the intermediate system may be a public or untrusted system. It would also apply to unidirectional transmission systems, including some command and control systems as well as systems that transmit to destinations operating under radio silence rules.

Finally, it should be noted that in some cases the communications system may employ some device such as an Automated Teller Machine to screen the messages being sent, allowing only the "good" messages through. But in this case the system (the ATM machine and the bank) and the user do not necessarily share common interests. The user may wish to ensure that his messages are kept secret, and the legitimate user may also be interested in assuring the end-to-end integrity of his messages. But the system, in this case the ATM machine, may also have a role to play in assuring that the user does not compromise the integrity of his own messages.

We should not try to satisfy both of these possibly diverging requirements through one mechanism. Instead, just as we sometimes use super-encipherment (for example using end-to-end

DES encryption to ensure writer-to-reader privacy, plus link encryption using classified algorithms to protect against an external threat), we should talk about super-authentication. That is, *if the system has a requirement to assure that messages are not modified after they exit a secure processing facility, then the system must independently provide that assurance without depending upon the user's mechanisms.*

3 A Quadratic Congruential MDC

Now that we have developed the rationale for the use of an MDC algorithm, we should certainly try to define a suitable implementation:

3.1 The Original QCMDC.

The original Quadratic Congruential Manipulation Detection Code (QCMDC) function proposed in the second paper in this series was defined as:

$$Z_0 = C = \text{MDC initial value}$$

$$Z_i = (Z_{i-1} + X_i)^2 \text{ modulo } N$$

$$\text{MDC} = Z_n,$$

where C, Z_i, and MDC are all 32-bit integers in two's-complement notation, and N was the Mersenne prime 2^{31}-1, chosen so that the modulo result would fit in a 32-bit word.

In order to prevent an attack against the MDC in the case of Output Feedback Mode (where both the text and the MDC could easily be changed), it was first proposed to make the first 32 bits of the message a secret seed, S, withheld even from the message originator, so that if the opponent attempted to attack his own message he would not know the secret seed and would therefore not be able to intelligently modify the MDC.

However, a variation of the under-determined knapsack attack of Coppersmith involving the taking of square roots modulo N and working backwards from the MDC in a meet-in-the-middle attack showed that the use of the secret seed, S, was not sufficient; and that either a secret quantity C would have to be introduced into the accumulator or the MDC would have to be extended to 80 bits or more.

When the QCMDC algorithm was first implemented on the 8087, some variations were also coded and tested which used an Exclusive OR operation (denoted ⊕ or XOR). These variations were intended to defeat Coppersmith's technique of working backwards taking square roots modulo P. Although these operations were felt at the time to increase the cryptographic strength of the algorithm by denying the attacker the opportunity to work backwards (by making the algorithm non-invertible), the additional operations were quite time consuming.

However, we concluded in the third paper that the MDC must be on the order of 128 bits long in order to foil the birthday problem attack in any case, and for that reason it was recommended that four separate iterations of the MDC algorithm be performed over the text resulting in a 124-bit MDC. It was therefore thought that Coppersmith's attack on the QCMDC would be defeated because of the difficulty of generating the requisite 2^{62} different variations. We then concluded that none of the variations on the basic QCMDC approach were necessary.

3.2 The Triple Birthday Attack

Ironically, one week before the publication of the third paper, Coppersmith[13] pointed out a weakness in a double-iteration DES signature scheme by Davies and Price which also applied (to a somewhat lesser degree) to the quadruple-iteration MDC scheme, as follows:

- Assuming the use of an arbitrary *invertible* function F(X,H) as a checksum function operating over the message $M = (M_1, M_2, \dots M_n)$, intermediate results $H_1, H_2, \dots H_n$ are produced from the relation $H_i = F(M_i,H_{i-1})$, or alternately from the inverse of F, $H_{i-1} = F^{-1}(M_i,H_i)$.

- During a precomputation phase, select some arbitrary n-bit quantity Z, which is going to be the value of $H_2, H_4, H_6, \dots ,H_{18}$. Then randomly select approximately 2^{36} values X, compute the values F(X,Z), and store these values. Then randomly select 2^{36} values Y, compute the inverse function $F^{-1}(Y,Z)$, and store those values as well. Then compare all of the Y values to all of the X values searching for a matching pair, using a sort and compare technique as required. This constitutes the first birthday problem. We expect to find 256 such matching pairs, and if not, we will examine a few more values of X or Y or both. Note that each such pair (X_i, Y_i) can be used as a message pair (M_3,M_4), (M_5,M_6), ..., or (M_{17},M_{18}) such that if $H_2 = Z$, $M_3 = X_i$, $M_4 = Y_i$ then $H_4 = Z$, etc.

- Given a message $M^* = (M_{19}, M_{20}, \dots , M_n)$, the chosen value of Z, and the 256 pairs (X_i, Y_i) obtained during the precomputation, our task is to select values of $M_1, M_2, \dots , M_n$ which will make H_{2n} a valid hash of $M = (M_1, M_2, \dots,M_n)$. We therefore find values of M_1 and M_2 such that $F(M_1,Z) = F^{-1}(M_2,Z)$ to put ourselves in a standardized position. This takes on the order of 2^{33} hashing operations and 2^{32} storage. This is the second birthday problem.

- Working backwards from H_{2n} (note that this requires the checksum function to be invertible), using the values $M_n, M_{n-1}, \dots , M_{19}$, we find the value of H_{n+18}, the value of the hash function on the *second* iteration. Finally, we make use of the precomputed pairs (X_i,Y_i). For each of the $256^4 = 2^{32}$ choices of the four pairs (X_i,Y_i) to be the values of (M_3,M_4), (M_5,M_6), (M_7,M_8), and (M_9,M_{10}), we compute the value of H_{n+10} that would result then do the same thing with the values of (M_{11},M_{12}), (M_{13},M_{14}), (M_{15},M_{16}), (M_{17},M_{18}), computing backwards from H_{18} to get a value for H_{10}. We again sort and compare these values as the third birthday problem. We expect one match, and the corresponding values of M_3 through M_{18} finish our task for a two-pass checksum process.

- The process could be extended to attack a triple-pass hash algorithm by constructing eight "super-pairs" consisting of M_{19} through M_{35} plus M_{36} through M_{52}, etc., up to M_{258}. Each

[13] Coppersmith, D., "Another Birthday Attack", Advances in Cryptology - CRYPTO '85 Proceedings, Lecture Notes in Computer Science, Vol. 218, Springer-Verlag, Berlin, 1986, pp 14-17.

super-pair would be manipulated during the precomputed phase to continue to produce the value of Z, even on the third pass. Only slightly more computation would be required, but obviously 258 blocks of the message M would be constrained, limiting the messages that could be attacked to fairly long ones. Finally, this process could be extended even further to attack a quadruple-pass hash algorithm by computing eight "super-dooper" pairs consisting of 512 blocks each, or a total of 4098 blocks.

The multiple birthday attack therefore serves to reduce the strength of an N-pass signature scheme from an apparent $2^{N*k/2}$ to an almost trivial $N*2^{k/2}$.

It is worth mentioning that the Coppersmith's attack also applies to attempts to extend the MAC of FIPS PUB 46 or ANSI X9.9 to 128 bits (in order to try to overcome Yuval's attack against the plaintext) by simply concatenating two or more MACs using two or more different authentication keys. The reason is that the MAC function, i.e., DES Cipher Feedback mode encryption, is invertible, and in addition the components are separable and individually too small to resist a birthday attack[14]. As a result, and *contrary to the advice in the second and third papers in this series, the 64-bit Message Authentication Code technique by itself cannot be considered sufficiently strong, and is not recommended* if there is any possibility that the originator may attempt to defraud the message recipient, or if a Trojan Horse could circumvent security controls through such a mechanism. In addition, the use of a MAC in certain command and control situations where the attacker may attempt to spoof computer-controlled equipment or processes is also not recommended.

In practice, the likelihood of all of these blocks of being substituted without being noticed may be remote, for in the case of the quadruple-iteration QCMDC routine this amounts to 16392 bytes that would have to be inserted in the text. However, in the previous papers we had committed ourselves to detecting even a single inserted, deleted, or manipulated bit, regardless of the amount of text and independent of any internal syntactical or semantic content. After all, if we were to rely solely on internal consistency checks to detect such manipulations we would first have to invent a suitable manipulation detection scheme!

It should therefore be observed that Coppersmith's triple-birthday attack will succeed against a multiple-iteration QCMDC routine if two conditions are true:

1. If the checksum function is *invertible*, so that it is possible to work both forwards and backwards to produce matching values in a birthday-problem attack.
2. If the checksum function is subject to *decomposition* into separate and independent elements, each of which is sufficiently small that the birthday-problem attack is feasible from the standpoint of computation time and storage. If the checksum function were to involve a 128-bit result that could not be broken down into something smaller, then the birthday attack would be infeasible because it would involve generating, storing, and comparing on the order of 2^{64} 128-bit checksums and 64-bit permutation indices, or about $8.8*10^{20}$ bytes of storage, or 5 quadrillion reels of 6250 bpi magnetic tape.

[14] This is not to say that a suitable 128-bit checksum could not be constructed using DES or some other 64-bit block cipher, but only to caution that the task is not nearly as trivial as it may appear at first glance.

In the case of the simple QCMDC routine (where $H_i = ((H_{i-1} + M_i)^2)$ modulo N), the addition of H_{i-1} and M_i makes the function technically non-invertible from the standpoint of exactly and uniquely reproducing the input F_{i-1} given some F_i, since the H_i is a function of two independent variables. But it is sufficient if the attacker can construct a value $Y_{i-1} = F^{-1}(X_i)$ which, when computed in the forward direction, will produce the desired result for H_i. To do this, note that (K*N + X) mod N = X. Therefore, multiply the modulus N by some variable K such that the result is a perfect square, and take the square root of the result. Then $Y_{i-1} = H_i$ - K*N, and the value of X_i that will satisfy this relation is SQRT(K*N) - Y_i.

This suggests a variation of the QCMDC routine that would involve XOR(s) or some other non-linear combining function that would not be susceptible to a square root attack. If in addition the routine involved all 128 bits of the text and all 128 bits of the MDC of the previous block, then neither of the two conditions would be true and the triple-birthday attack would therefore be defeated. However, as the indefatigable Dr. Coppersmith pointed out, this is not necessarily a trivial task.

In order to make the MDC function non-invertible it is necessary to introduce a history function, i.e., some value that would not yet be known when working in the backwards direction, calculated in some non-linear manner so that the square root attack will not work. In addition, it appears necessary to incorporate multiple references to both the text to be authenticated and to the previous MDC result, so that the only value that would satisfy the forward relationship is the proper one. Not only must each bit of the checksum function be a function of all of the bits in the full 128-bit text block together with all of the bits in the MDC of the previous block, but additional dependencies should be introduced to ensure that the function is not just minimally dependent on those bits but is over-constrained instead.

Finally, as stated previously, the MDC function must produce a value on the order of 128 bits in length in order to defeat the various birthday attacks against the text itself.

3.3 The New, Improved QCMDCV4 Algorithm

The following algorithm, dubbed the Quadratic Congruential Manipulation Detection Code, Version 4 (QCMDCV4) for brevity, is proposed to satisfy these requirements:

Consider a 128-bit (16 byte) block of text, divided into four 32-bit words, $T_1, \ldots ,T_4$. For reasons that will be explained later, we will be operating on a 31-bit subset of each of those 32-bit words which consists of the sign bit and the low-order 30 bits, i.e., $T^*_i = T_i$ AND BFFFFFFF. In addition, we will define a 30-bit fifth component, T^{**}, consisting of the 6 high-order bits of T_1 (with the 6 bits shifted right two bits and 2 leading zero bits introduced on the left or most-significant-bit position), concatenated with the high order 8 bits of T_2, T_3, and T_4, to make a 32 bit word with two high order zero bits.

Let the 128 bits of the MDC result (obtained from the previous block of text) also be divided into four 32-bit integer components M_1, M_2, M_3, M_4; and let the 32-bit components of the new MDC result be designated as M^*_i.

Finally, define a set of moduli $N_1 ... N_4$, consisting of the four largest prime numbers less than the maximum 32-bit integer, namely 2147483629 (2^{31}-19), 2147483587 (2^{31}-61), 2147483579 (2^{31}-69), and 2147483563 (2^{31}-85).

Then calculate:

$$\begin{array}{lll} M^*_1 = & [& (M_1 \oplus T^*_1) - \\ & & (M_2 \oplus T^*_2) + \\ & & (M_3 \oplus T^*_3) - \\ & & (M_4 \oplus T^*_4) + T^{**}]^2 \bmod N_1 \end{array}$$

$$\begin{array}{lll} M^*_2 = & [& (M_2 \oplus T^*_1) - \\ & & (M_3 \oplus T^*_2) + \\ & & (M_4 \oplus T^*_3) - \\ & & (M^*_1 \oplus T^*_4) - T^{**}]^2 \bmod N_2 \end{array}$$

$$\begin{array}{lll} M^*_3 = & [& (M_3 \oplus T^*_1) - \\ & & (M_4 \oplus T^*_2) + \\ & & (M^*_1 \oplus T^*_3) - \\ & & (M^*_2 \oplus T^*_4) + T^{**}]^2 \bmod N_3 \end{array}$$

$$\begin{array}{lll} M^*_4 = & [& (M_4 \oplus T^*_1) - \\ & & (M^*_1 \oplus T^*_2) + \\ & & (M^*_2 \oplus T^*_3) - \\ & & (M^*_3 \oplus T^*_4) - T^{**}]^2 \bmod N_4 \end{array}$$

Several features of this algorithm should be noted. First, each of the 16 different XOR combinations is unique. Second, even if a significant amount of the text contains all zeroes (with the result that the XOR does nothing), the alternating signs for the M_i and T^{**} components operate in such a manner that the contribution of the various terms will be different in each case. Finally, the M^*_i values are introduced into the computation of the subsequent components as soon as they are available, so that there is a great deal of inter-dependency and mixing. As a result, each 32-bit component of the MDC result is an over-constrained function of all of the text and all of the prior MDC.

The previous papers had proposed a constant value for the modulus, N, equal to the Mersenne prime 2^{31}-1 (2147483647), for all four of the 32-bit M^*_i results. But as Don Coppersmith pointed out when reviewing a draft of the current procedure, because 2^{31}-1 is the largest number that can be contained in a four byte integer in two's complement form, XORing the hexadecimal bit-string 80000001 has the effect of inverting the sign and the low order bit, which can be the equivalent of adding or subtracting the modulus. As a result, even when the intermediate sum is squared, the division by the 2^{31}-1 modulus frequently produces no change in the result, depending on the sign of the T_i and whether a carry would be required, and a modification to the text could thereby escape detection.

Coppersmith proposed picking up the text only 24 bits at a time to avoid this problem, using additional iterations to get back to around 128 bits. In an attempt to overcome this problem without the overhead of an additional iteration, the four different primes for the moduli N_i

were introduced, all of them smaller than 2^{31}. However, it was found that if the text consisted of one 32-bit word of random bits and three words of zeroes, then in about 10% of the cases it was possible to either add or subtract the value of the first modulus and have the change go undetected in the corresponding 32-bit word of the MDC result. Although the use of four different values for the moduli means that the substitution does affect the remaining 3 words, or at least 96 bits, it was felt that the full 128-bit strength should be preserved.

For this reason, only 30 bits plus the sign bit of each 32-bit word of text is used in forming the intermediate sum. Since the moduli are all greater than 2^{30}, it is impossible to add or subtract the modulus from the text without detection. The final addition or subtraction of T^{**} ensures that all of the bits in the text affect all of the bits of the result.

One further improvement is possible. Because of the squaring operation, each 32-bit MDC component will be positive, producing a 124-bit result. But we can calculate the parity of the intermediate MDC result, just prior to the multiplication, and then change the sign of each 32-bit result if the parity is even.

Finally, because the algorithm operates on 16-byte blocks, it is necessary to somehow differentiate between a text string that is say 1 byte long and one that consists of the same byte extended with up to 15 bytes of zeroes. For that reason the last few bytes (less than 16), if any, are moved to a 16-byte buffer, the rest of the buffer zeroed, and the MDC algorithm executed N+1 times on that same buffer, where N is the number of the last few bytes. N+1 is used instead of N, because a block that is 16 bytes long has to be processed once, and therefore a 1 byte block has to be processed twice in order to be distinguished from the previous case. If improved performance is needed, the length code of the text can be prefixed to the text, and the size of the buffer extended to be an exact multiple of 16 bytes. This technique *must* be used if it is necessary to deal with text strings that are not multiples of 8 bits in length.[15]

In order to avoid a strong correlation between the text and the MDC result in the case where the text is very sparse (contains mostly zero bits), it is desirable to use different values for the starting values of M_i. For purposes of standardization the values 141421356, 271828182, 314159265, and 57721566 are suggested.

4 Implementation Considerations

The QCMDCV4 algorithm has been implemented and tested on the IBM PC and AT microcomputers and the Compaq 286 Portable, and should run correctly on any similar machine which uses the Intel 8088, 8086, 80188, or 80286 CPU chip in combination with the 8087 or 80287 Numeric Data Processor chip. The 8087/80287 is used to significantly speed up the calculation of the various arithmetic operations, in particular the *division modulo* the large primes.

[15] It may be worth mentioning that the ANSI X9.9-1986 authentication standard and the definition of the MAC in FIPS PUB 46 do not take this problem into account, and therefore do not differentiate between a short message (one that is not a multiple of 8 bytes in length) that must be padded with zeroes, and one that is a multiple of 8 bytes in length and happens to contain zeroes at the end. Although binary zeroes would be interpreted as ASCII null characters and would not be confused with the ASCII "0" (hexadecimal 30) character in coded text, formatted binary information is allowed by paragraph 5.1 of that standard, which does not specify that a length indicator field must be used. The confusion therefore could occur in this case.

During the calculations the results are kept in IEEE Binary Floating Point 80-bit Temporary Real format with a 64-bit mantissa, and T_i and M_i are in Intel 32-bit integer (IBM/Microsoft Pascal INTEGER4) format. (Note that the Intel format loads and stores register contents in "reversed" order, i.e., with the low order byte coming first in memory, so that the text bytes are processed in the order 4, 3, 2, 1, 8, 7, 6, 5, etc.)

In the worst case, the total resulting from the alternating sign terms could range from -2^{33} to $2^{33}-4$, in which case the squaring operation would produce a value as large as 2^{66}. Because the operation is carried out in floating point an overflow cannot occur, but a number that large cannot be represented in the 64-bit mantissa without loss of precision. If the 8087/80287 control word status were set to enable the precision interrupt then an interrupt would occur in that event, but the normal Pascal setting is to disable such interrupts. The result in the normal case will therefore be to round up or down to the nearest even value as appropriate (assuming the normal setting for the rounding mode), and discarding up to four low order bits of the sum. It should be noted that for precision loss to occur, the signs of the 32-bit result of the XOR must be +, -, +, -, to match the order of operations. As a result, it would be extremely unlikely for a loss of precision to occur on all four of the 32-bit intermediate result computations because of the way the text is cycled through the algorithm. In addition, if the intermediate result is viewed as the sum $2x + y$, where x represents the 31 high order bits and y the two low order bits, then the square is $4x^2 + 4xy + y^2$. Therefore, even though the low order y^2 bits are dropped after the multiplication this does not mean that the low order bits of the original quantity are ignored, since they affect the mid-square (4xy) component of the result. For this reason it is not possible for the low order bit or bits of one or more of the 32-bit words of text to be changed without causing a change in all 128 bits of the result.

The 8087/80287 FPREM instruction computes an exact remainder by successive subtractions the way division is done by hand, instead of using the more usual technique of dividing, rounding, multiplying, and subtracting from the original. The FPREM instruction is as fast as a divide, and is guaranteed to be accurate, without any roundoff. However, because the modulus is slightly less than 2^{31} and the maximum value of the result after the squaring operation is 2^{66}, the FPREM operation is not guaranteed to be completed in one operation (since the difference in magnitude between the dividend and the divisor may be larger than 2^{64} and FPREM shifts at most 64 bits in one operation), but it will always be complete in two operations. For this reason, the 8087/80287 condition code is tested after each FPREM and an additional FPREM performed if necessary.

In order to produce the fastest possible implementation, the XORs and other CPU instructions are executed in parallel with the coprocessor addition, subtraction, multiplication, and FPREM operations whenever possible. The FWAIT instructions necessary to ensure that the coprocessor has finished with its computations before the CPU reads the results are delayed as long as possible to permit the maximum possible overlap. Although the original version was coded using a macro that was invoked four times for the four different iterations within one block, in the final version the code was "unwound" and hand-optimized to permit maximum overlap.

On an IBM-PC with an 8088 & 8087 and a 4.77 MHz clock, the time to MDC check 1,000 512-byte blocks was 43.5 seconds, or 1359.5 microseconds per 16 bytes. This corresponds to 94.2 kilobits per second. By comparison, the time for the fastest known software implementation of DES for the PC is 2801 microseconds per 8 bytes for the PC (22.8 Kbps, or 171K bytes per minute). With an 80287 speedup kit (consisting of an 8 MHz 80287 with its own clock crystal on a plug-in daughter-board) installed in an IBM AT with the standard 6 MHz 80286, the same test took 813.6 microseconds for 16 bytes (157.3 Kbps), or 1.18 megabytes per minute, compared to the DES time of 933 microseconds per 8 bytes. We are currently awaiting the availability of the new Intel 80386 CPU together with the 80387 coprocessor to time that configuration. We expect to recode the algorithm to take advantage of the new 386/387 instructions, and anticipate that the result will run about 4 times faster than on the IBM AT. Depending on the clock speeds of the processors involved, then, the 128-bit MDC technique is anywhere from 4.6 to 8.1 times faster than computing two independent 64-bit Message Authentication Codes in software using the fastest known software DES implementation for the IBM PC or AT.[16] From a human factors standpoint, this means that the entire contents of a floppy disk (362K bytes) can be authenticated to the most stringent standards in less than 15 to 30 seconds on current microprocessors, without benefit of any special cryptographic hardware.

4.1 MDC Test Program

The following program, written in IBM/Microsoft Pascal for the IBM PC, can be used to verify the proper operation of the QCMDCV4 algorithm:

```
{$TITLE:'CHECKMDC' - Verify MDC algorithm.}
{$FLOATCALLS- (Generate native 8087/80287 code.)}

PROGRAM checkmdc(input,output);

TYPE
 checksums=      ARRAY[1..4] OF INTEGER4;

VAR [PUBLIC]
  text: PACKED ARRAY[1..33] OF CHAR;
  text_p:        ADSMEM;
  n_bytes:       WORD;
  result:        checksums;
  i,j:           INTEGER;

VAR [EXTERN]
   mdc_name:     PACKED ARRAY[1..8] OF CHAR;
                 {"QCMDCV4 "}

CONST
  mdc_init =     checksums(
                 141421356,
                 271828182,
                 314159265,
                 57721566);
  check =        checksums(
                 -1900412449,
                 676867420,
                 -689076088,
                 1333643940);
```

[16] In addition, two independent 64-bit MACs are not believed to be nearly as secure as a single 128-bit MDC.

```
PROCEDURE mdc(  text_ptr:ADSMEM;
                n_bytes:WORD;
           VARS result:checksums);
           EXTERN;

BEGIN;

WRITE(output,
  'Verifying MDC routine (',
  mdc_name,')...');

FOR i:= 1 TO 33 DO text[i] := CHR(0);
text[1] := CHR(1);
text_p := ADS text;
result := mdc_init;

FOR i:= 1 TO 50 DO
  BEGIN;
    IF i<34 THEN n_bytes := WRD(i)
    ELSE n_bytes := 32;

    mdc(text_p,n_bytes,result);

    FOR j:= 32 DOWNTO 1 DO
      text[j+1] := text[j];

    text[1] :=
    CHR(LOBYTE(LOWORD(result[4])));

    END;

IF result[1]=check[1] AND THEN
   result[2]=check[2] AND THEN
   result[3]=check[3] AND THEN
   result[4]=check[4]
  THEN WRITE('OK.')
  ELSE
    BEGIN;
       WRITE('MDC is INCORRECT!');
       WRITELN(result[1],result[2],
               result[3],result[4]);
     END;
WRITELN;

END.
```

5 Summary and Conclusions

Several architectural justifications have been presented an authentication algorithm which does not require a traditional crypto "black box" approach using secret cryptographic keys, with all of the key management difficulties that entails. In particular, the relatively common practice of using link encryption for secrecy at the OSI Data Link layer and implementing end-to-end authentication at the Presentation Layer would profit from "keyless", non-cryptographic means of authentication that could be easily implemented in both PCs and general-purpose main-frame computers.

The need for a checksum on the order of 128 bits in length was reaffirmed, both in the case of two mutually suspicious, potentially deceitful users where one may attempt to defraud the other, and in the command and control case where the attacker may have an almost unlimited ability to attempt to spoof the system. Contrary to the author's previous position, it was concluded that the 64-bit Message Authentication Code (MAC) approach of FIBS PUB 46 cannot be considered sufficiently strong in the case where the originator of a message may attempt to defraud the recipient, as well as in some command and control and multi-level security situations.

The MAC checksum technique used by ANSI X9.9-1986 is viewed as particularly unfortunate, both because of the inadequate 32-bit length and because no provision was made to distinguish between short block that was padded and a block that is a multiple of 8 bytes that happens to end with the same characters.

Coppersmith's Triple Birthday attack as it applied to the original QCMDC algorithm was summarized, and it was concluded that in order for that attack to be defeated it was necessary to ensure that the checksum function is not invertible, and that the length of the checksum be on the order of 128 bits in length.

The QCMDCV4 algorithm was described, which uses XORs plus a history function to ensure that the function is not invertible. The function computes a 128-bit result that is an over-determined function of 128 bits of the text and the 128-bit MDC result of the previous text block than cannot be decomposed. A "birthday attack" against the QCMDCV4 result cannot succeed, because of the enormous number of variations that would have to be computed, sorted and compared. In order to ensure that a message that is not an even multiple of 128 bits can be distinguished from the same message extended with zeros, the algorithm is executed N + 1 times on the last buffer, which contains the last N bytes of data extended with zeros.

The QCMDCV4 algorithm is recommended for use in microcomputer and main-frame applications where encryption will be provided separately and it is desirable not to have to replicate the encryption function for authentication. It is also suitable for use in combination with a public-key algorithm when implementing a digital signature function to protect against fraud.