Implicit Data Integrity: Protecting User Data without MACs

Michael Kounavis, David Durham, Sergej Deutsch and Saeedeh Komijani
Intel Labs, Intel Corporation, 2111, NE 25th Avenue, Hillsboro, OR 97124, U.S.A.

Keywords: Data Integrity, Implicit Integrity, Pattern Detectors, Entropy, Entropy Index, Symmetric Encryption.

Abstract: We address the problem of detecting data corruption, without producing, storing or verifying mathematical summaries of the content, as it is typically done today. Data corruption may be either due to natural means or due to the malicious modification of content by some attacker or malware. Today, data corruption detection is supported by producing and using security metadata such as Message Authentication Codes (MACs), Integrity Check Values (ICVs), checksums etc. The methodology we study, called 'implicit data integrity' avoids the use of such metadata. It supports the detection of corruption in a novel way based on the observation that regular unencrypted user data typically exhibit patterns. When some encrypted content becomes corrupted and is decrypted, it may no longer exhibit patterns. It is the absence or presence of patterns in decrypted content which denotes whether some content is modified or not. We present a number of pattern detectors and algorithms which can successfully support implicit data integrity at quantifiable security levels. We also demonstrate that our patterns and algorithms can characterize the overwhelming majority of client and server workload data. We present security analysis and performance results coming from over 111 million representative client workload cache lines and 1.47 billion representative server workload cache lines. We also present synthesis results showing the efficiency of the hardware implementations of some of our algorithms.

1 INTRODUCTION

1.1 The Concept of Implicit Data Integrity

We address the problem of detecting data corruption without producing, storing or verifying mathematical summaries of the content. By 'mathematical summaries of the content' we mean integrity metadata such as Message Authentication Codes (MACs), Integrity Check Values (ICVs), Cyclic Redundancy Codes (CRCs) or checksums that can be used for detecting whether some content has been unintentionally or maliciously modified. Content can be modified by natural means (e.g., due to noisy channels or interference), by software bugs (e.g., due to bugs corrupting memory regions) or intentionally (e.g., due to malware or man-in-the-middle attacks).

Today, the standard way of supporting data integrity is by using integrity metadata. Integrity metadata can be cryptographically strong, as in the case of MACs (SHA-256, 2012), (SHA-3, 2016), (HMAC, 2008), (KMAC, 2016), or weak(er) but efficient as in the case of checksums, CRCs, or Reed-Solomon codes. Furthermore, some types of metadata may support error correction, whereas other types of metadata may not. What is common in all such metadata is their associated latency, storage and communication bandwidth overheads.

Overheads are due to the unavoidable content expansion resulting from producing and storing the metadata. In storage systems, for instance, extra space may be required for storing and accessing ICVs, Reed-Solomon codes or MACs. The cost of such metadata in terms of extra storage space is typically non-negligible. In computing systems with cryptographic protection of memory (McKeen, 2013) each cache line needs to be protected by a separate MAC value. In this way, a separate MAC value needs to be also read in every data read operation. This design choice may result in wasting significant memory access bandwidth resources in a processor, as each data read operation may need to be realized as two memory read operations in the worst case. In computer communication systems that need to reliably transmit data from a source to a destination end-point, integrity metadata need to be transmitted as well. The transmission of such metadata, whether checksums, CRCs, or MACs may further consume significant communication bandwidth resources.

Kounavis, M., Durham, D., Deutsch, S. and Komijani, S.
Implicit Data Integrity: Protecting User Data without MACs.
DOI: 10.5220/0006905105430552
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, pages 543-552
ISBN: 978-989-758-319-3

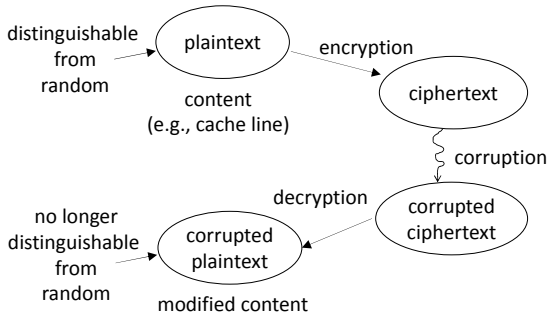

Figure 1: Implicit Data Integrity.

In this paper we study an alternative methodology that avoids the use of some security metadata altogether. he methodology we study employs pattern-based algorithms in order to support data corruption detection without content expansion. The main idea, which we study in this paper comes from the work by Durham et. al. (Durham, 2013, 2015) and is illustrated in Figure 1. If some content exhibits patterns (i.e., has low entropy), then such content can be distinguished from random data. Let's consider that this content is encrypted, as shown in the figure, where the encryption algorithm is a wide block cipher and a good pseudo-random permutation. Thus, it can successfully approximate a random oracle (Hall, 1998), (Gilboa, 2015). The cipher text which is produced in this way is no longer distinguishable from random data, under certain reasonable assumptions about the adversary. Any corruption on the cipher text results in a new cipher text value, which is different from the original one. Furthermore, any decryption operation on this new cipher text value results in a corrupted plaintext value which is different from the original one as well. As decryption is the inverse operation of encryption, the decryption algorithm also approximates a random oracle. Because of this reason, the corrupted plaintext value is also indistinguishable from random data with very high probability.

From a system realization stand-point, the corrupted plaintext is indistinguishable from random data due to the fact that wide block ciphers, such as (Ferguson, 2009), typically perform strong mixing of their input bits. Due to an 'avalanche effect' associated with the decryption oracle, even a single bit change in the cipher text affects all bits of the decrypted plaintext. Therefore, checking the entropy of the result of a decryption operation can be a reliable test for detecting corruption for some data. We refer to such methodology as 'implicit data integrity' or just 'implicit integrity'.

1.2 Challenges

One of the main challenges in building systems that are based on the principles of implicit integrity is how to define 'high' or 'low' entropy. It is not straightforward how to determine that some content's entropy is 'low enough' or 'high enough' so as to safely deduce that the original content has not been corrupted. The standard definition of entropy may not be easily applicable to corruption detection because message sizes may be small (e.g., messages may be memory cache lines of 512 bits), or the sizes of symbols where deterministic behavior is demonstrated may vary significantly (e.g., deterministic behavior may be observed over sets of nibbles, bytes, words or double words). A good corruption detection system must use algorithms that, on the one hand, characterize the overwhelming majority of the user data as 'not random' or 'of low entropy'. On the other hand, the same algorithms when applied to corrupted plaintext values must characterize such values as 'random' or 'of high entropy' with probability very close to 1. Building a system that meets these requirements is hard. In this paper we present a solution that successfully addresses this problem. Our solution is not the only one that has been proposed in this space (Durham 2013, 2015 2016). However, our solution gives the best results when compared to alternatives. In fact, it is the most optimal solution known to us and, according to our results, state-of-the-art for the aforementioned problem.

1.3 Contributions of This Paper

The paper makes two contributions toward building systems based on the principle of implicit integrity. A first contribution is an experimentally derived set of pattern detectors which characterize the overwhelming majority of unencrypted uncompressed user data and can be used together with wide block ciphers for building practical working systems supporting implicit integrity. A second contribution of this paper is a proposal of a new entropy measure, called entropy index, which is more appropriate for small size quantities (e.g., 512 bits, 1024 bits) and can be used for quantifying the security levels offered by pattern detectors.

A related approach is described in reference (Durham, 2013) which introduces the idea of implicit integrity. This approach avoids specifying how pattern detectors operate. We significantly expand upon the idea of reference (Durham, 2013), providing new components for implicit integrity. Another

related proposal is described in reference (Durham, 2016). In this work, the content of a small size entity (e.g., a cache line) is deemed 'not random' if it demonstrates 4 or more 16-bit words equal to each other. Compared to this solution, our solution is more generic. It further increases the percentage of client and server workload data that can be protected using the implicit integrity methodology significantly. For memory workloads, the observed increase is from 82% to 91% for client data, and from 78% to 84% for server data, while supporting a level of security of 31 bits. These results are collected from 111 million representative client workload cache lines and from 1.47 billion representative server workload cache lines.

1.4 Other Considerations

From this discussion, it becomes evident that not all data can be protected using implicit integrity. We envision implementations that protect the overwhelming majority of user data that exhibit patterns using the implicit integrity methodology and the remaining data using standard techniques. Furthermore, we show that implicit integrity can be supported at various security levels, which are as measurable as MAC lengths are. In one example, for instance, the security level supported by our solution can be at 31 bits. By '31 bits' of security we mean that the probability of having a corrupted plaintext value being characterized as 'of low entropy' is $2^{-31+\epsilon}$ for some small security margin ϵ. The security level offered is associated with an 'entropy index' value characterizing the input data, which is defined in the next section of the paper. The higher an entropy index value is, the more difficult is for an attacker to corrupt some data. All security considerations are with respect to an adversary who performs on-line attacks by corrupting cipher text values. By 'on-line attacks' we mean attacks where the detection of even a single corruption event exposes an attack. Finally, we note that the solution described in this paper is applicable to any type of data, including memory, data communication data and storage data. The following description is independent of the data types where our solution applies.

The paper is structured as follows. In Section 2 we provide an overview of the algorithms discussed in this paper. In Section 3 we discuss related work and contrast our approach to the known state-of-the-art. In Section 4 we provide details on our approach. Finally in Section 5 we provide some concluding remarks.

2 PATTENS AND ALGORITHMS FOR IMPLICIT INTEGRITY

2.1 Experimentally Derived Pattern Detectors

The two contributions of this paper are illustrated in figures 2 and 3 respectively. The first contribution, shown in Figure 2, is a collection of experimentally derived pattern detectors that support implicit integrity using a set of thresholds T_1, T_2, …, T_8. This is the simplest of our contributions, and a rather straightforward algorithm to start the exploration of this space. The second contribution of figure 3 is an algorithm that performs pattern checks similar to those of the algorithm of figure 2, but using entropy index values.

The algorithm of Figure 2, called Extended Pattern Matching (EPM), employs many different styles of pattern checks. The security levels associated with these pattern checks are discussed in Sections 2.2 and 4. One type of pattern checks detects entities among the input data that are equal to each other. Entities can be nibbles, bytes, words (16-bit) or double words (32-bit). Another type of pattern checks concerns entities among the input data which are not only equal to each other, but are also placed in continuous index positions. This second type of pattern checks is not necessarily the same as the first one. For example, one can associate these two types of pattern checks with different thresholds and, by doing so, build two different pattern detectors. Yet another type of pattern checks detects entities that take special values. Special values are values that are frequently encountered in regular user data but are infrequently encountered in random or corrupted plaintext data. For example, in memory cache lines obtained from client data workloads, a high percentage of bytes take the values of 0x00 or 0xFF.

A last type of pattern checks detects entities, the value histogram of which demonstrates a sum of n highest entries (i.e., frequencies) being higher than a threshold. The intuition behind this type of pattern check is that there are several types of input messages, the content of which is not as random as that of encrypted data, but also does not demonstrate patterns at the byte or word granularity. One example of such content is media data, where nibble values may be replicated, but data do not demonstrate significant byte or word replications. Experimental studies over 111 million client cache lines and 1.47 billion server cache lines have shown that there are millions of cache lines demonstrating a limited set of

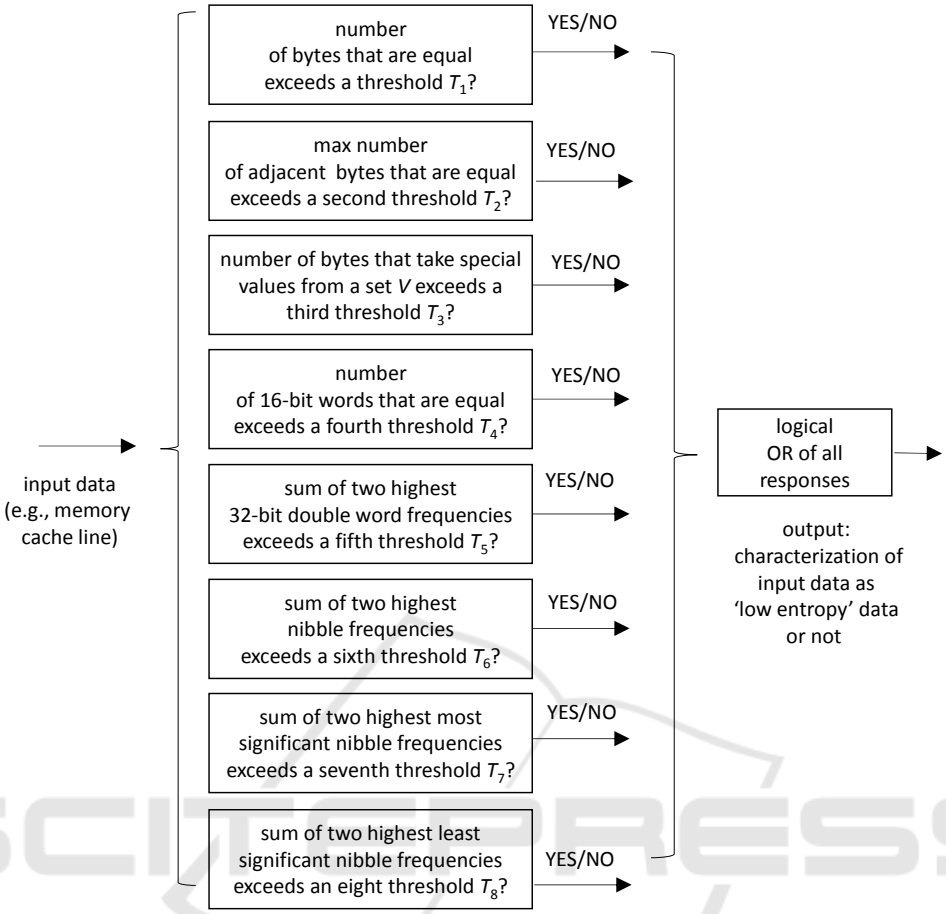

Figure 2: EPM that uses threshold values.

byte equalities, but a substantially higher number of nibble equalities. A pattern check that computes the maximum number of nibbles that are equal to each other is not appropriate in this case, as we have shown that such pattern check is limiting both in terms of the percentage of user data it is applicable to, and in terms of the security level it offers. An alternative pattern check that computes whether the sum of the n highest nibble frequencies exceeds a threshold is more efficient. If the input data consist of cache lines, this pattern check works best for $n = 2$. By checking whether the sum of the two highest nibble frequencies exceeds a threshold, a more flexible pattern detector can be built, which on the one hand encompasses significantly more regular user inputs, and on the other hand is associated with an event that is infrequent among random data.

The types of pattern checks outlined above can be applied in many different types of data values and can be further combined resulting in stronger pattern detectors. The diagram of Figure 2 illustrates one example where 8 pattern checks are employed. The following computations are performed in order to be determined whether the resulting numbers or frequencies exceed a threshold: (*i*) the computation of the maximum number of bytes being equal; (*ii*) the computation of the maximum number of adjacent bytes being equal; (*iii*) the computation of the maximum number of bytes that take special values from a set V; (*iv*) the computation of the maximum number of words being equal; (*v*) the computation of the sum of the two highest double word frequencies; (*vi*) the computation of the sum of the two highest nibble frequencies; (*vii*) the computation of the sum of the two highest most significant nibble frequencies; and (*viii*) the computation of the sum of the two highest least significant nibble frequencies.

The resulting numbers/frequencies are compared against thresholds T_1, T_2, …, T_8 and the responses form a vector of eight Boolean values. These values undergo a logical OR operation. The result of this logical OR operation is the final response on whether the input is of low entropy or not. Essentially, the algorithm of Figure 2 checks whether there exists at

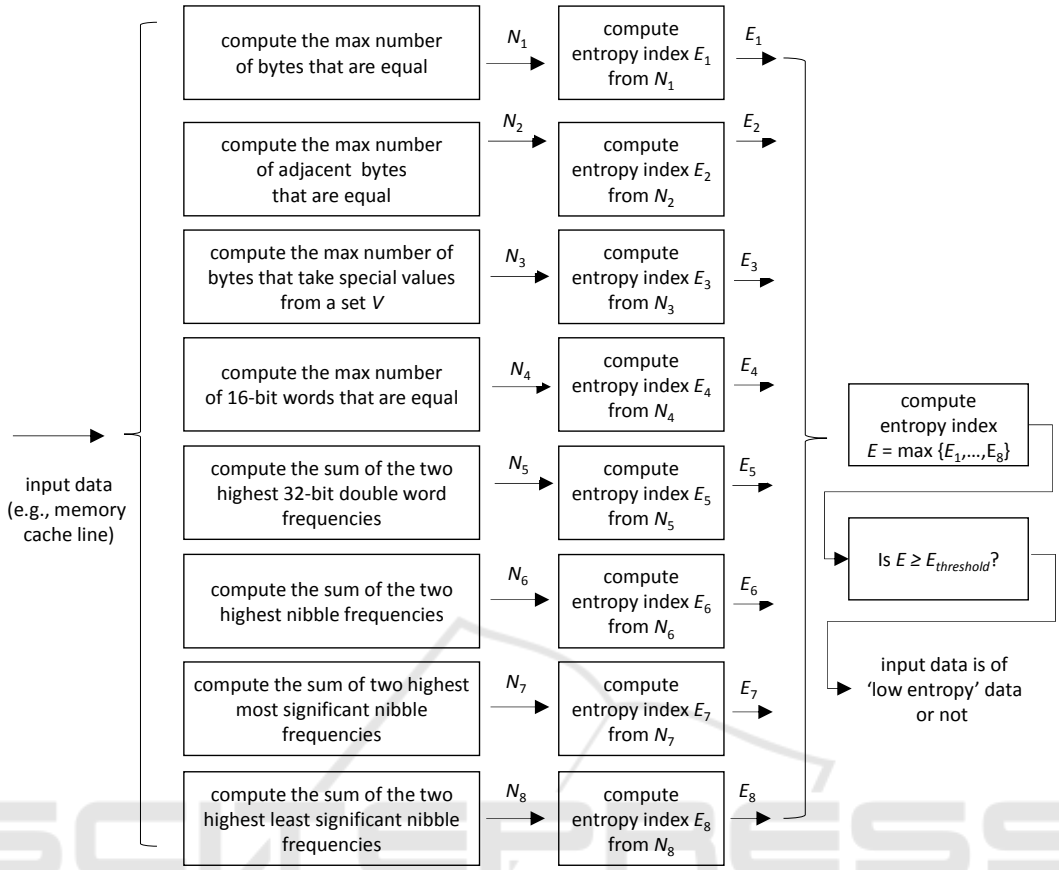

Figure 3: EPM that uses entropy index values.

least one pattern check from among the set employed, according to which the observed entities which exhibit the pattern exceed a threshold. If such pattern check exists, the input is characterized of low entropy, otherwise it is not.

2.2 The Concept of the Entropy Index

Our second contribution is illustrated in Figure 3. This is an extension of Extended Pattern Matching that does not require the use of thresholds. Instead, the algorithm computes the number of entities that exhibit a pattern in some data and converts this number into an 'entropy index' value. In what follows we define the concept of an entropy index. Let's consider a pattern p of type π (e.g., maximum number of bytes being equal). If some input data x of fixed length L demonstrates pattern p and exactly N entities from x exhibit this pattern (e.g., the maximum number of bytes being equal is exactly N) we denote this fact as:

$$x \in p(\pi, N) \tag{1}$$

We define an entropy index value E associated with the pattern type π and entity number N as the negative logarithm of the probability that $x \in p(\pi, N)$ given that x is random (e.g, x is obtained from the output of a random oracle).

$$E(\pi, N) = -\log_2 \text{Prob}[\, x \in p(\pi, N) \mid x \leftarrow \text{trunc}_L(x_0), x_0 \leftarrow \$\,] \tag{2}$$

where by $\text{trunc}_L()$ we mean a function that truncates its input returning only L bits of it.

According to the definition above, the probability of seeing the pattern $p(\pi, N)$ in a random data value x is equal to 2^{-E}. Entropy index E is measured in bits. Furthermore, the expected number of random values we need to inspect until we find one that demonstrates the pattern $p(\pi, N)$ is 2^E. As a result, the entropy index E associated a pattern type π and an entity number N is also equal to the logarithm of the expected number of random values we need to inspect until we find one value x such that $x \in p(\pi, N)$.

The Extended Pattern Matching variant of Figure 3 employs the same pattern checks as the algorithm of Figure 2 with one exception. The pattern checks do

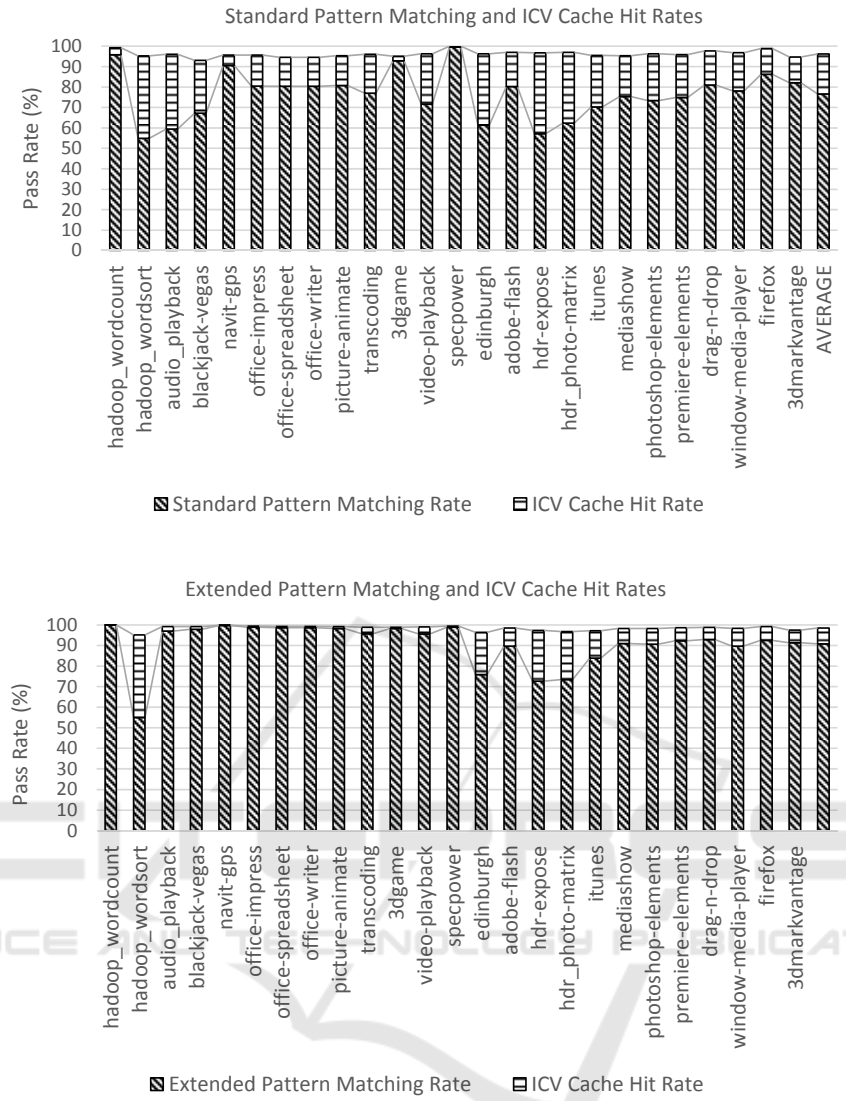

Figure 4: Pass rates of the SPM and EPM algorithms.

not return Boolean responses. Rather, the pattern checks return the actual numbers of the entities that exhibit patterns $N_1, N_2, \ldots, N_8$. Each of these numbers is converted to an entropy index value. The computation of every entropy index value depends on the type of the pattern check used and on the number of entities that exhibit the pattern in the data. In Section 4 we provide examples of how entropy index values can be computed for different types of pattern checks. The entropy index values $E_1, E_2, \ldots, E_8$ obtained this way undergo a stage that selects the maximum of these. If the maximum entropy index E exceeds a threshold, then the input is characterized as demonstrating low entropy, otherwise it is not.

The algorithm of Figure 3 essentially searches for the rarest of the patterns that are exhibited by the input data. The rarest of the patterns is the one that appears with the smallest probability among random data and is associated with the highest entropy index value. As the algorithm converts numbers $N_1, N_2, \ldots, N_8$ into entropy index values $E_1, E_2, \ldots, E_8$, the algorithm does not need to directly operate on thresholds that characterize the data entities exhibiting the patterns. Instead, the algorithm of Figure 3 operates on a single entropy index threshold, which has practical significance. This threshold reflects the expected number of efforts required by an adversary in order to produce the rarest of the patterns considered in the figure by corrupting cipher text data. Such probability is associated with the highest entropy index value.

3 RELATED WORK

Message authentication is typically accomplished using cryptographic algorithms that involve hashing (SHA-256, 2012), (SHA-3, 2016), symmetric (AES, 2001), or asymmetric encryption. Today's techniques require additional storage or bandwidth for storing a mathematical summary of the message which is authenticated. An alternative approach is implicit integrity, which is first described in reference (Durham, 2013). Reference (Durham, 2013) introduces the idea of implicit integrity without specifying how pattern detectors are designed and operate. The degree of success of this approach depends on the design and implementation of the pattern detectors utilized. Similar to implicit integrity is also the idea of 'Robust Authenticated Encryption' (RAE) (Hoang, 2015), which typically involves some message expansion. In this approach, some redundancy λ is typically added to the message which is being protected.

A related proposal described in reference (Durham, 2016), concerns a pattern detector that detects whether 4 or more 16-bit words are equal to each other in a cache line. Compared to this solution, ours is more general, avoids the use of fixed thresholds, and increases the percentage of data that are protected with implicit integrity significantly. In memory systems, such increase is from 82% to 91% on client data, and from 78% to 84% on server data, while supporting the same level of security (31 bits). These results are collected from 111 million representative client workload cache lines and from 1.47 billion representative server workload cache lines. Pass rate comparisons between the solution of reference (Durham, 2016) and our solution are shown in Figure 4. By 'pass rates' we mean the percentages of data that demonstrate specific patterns such as those of Figures 2 and 3.

In the figure, we refer to the solution of reference (Durham, 2013) as Standard Pattern Matching (SPM). Our solution is referred to as Extended Pattern Matching (EPM). The pass rates for many different client workloads are shown in the figure. As is evident from the figure, there are many typical client workloads (e.g., Microsoft® Office, transcoding, video player) the pass rates of which range between 75%-80% using Standard Pattern Matching. These pass rates are boosted to 98% when Extended Pattern Matching is employed. Overall, the average pass rate with Standard Pattern Matching is 80%. The average pass rate with Extended Pattern Matching is 91%. For server data, the corresponding pass rate values are 78% and 84% respectively.

Figure 4 also shows pass rates for data that are not protected using the implicit integrity methodology but for which Integrity Check Values (ICVs) are stored in a quickly accessible cache unit. It is assumed that in addition to the implicit integrity methodology, an ICV cache unit is employed to protect those cache lines which do not demonstrate patterns. A 4 KB ICV cache unit is used for the charts of Figure 4. Average pass rates for the two algorithms are 97% and 99% respectively in this case.

4 SECURITY ANALYSIS AND IMPLEMENTATION

In our work we assume that the security levels supported by our algorithms are significantly smaller than the sizes of the input messages. For on-line attacks such assumption is not limiting. Neither the security offered is limiting, as in on-line attacks the adversary has only one chance to succeed. An unsuccessful attack exposes the adversary, resulting in a possible re-encryption of the data with a new key. For example, for memory cache lines of 512 bits we can safely support security levels up to 64 bits using our methodology. The reason why is because attackers do not in reality attack ideal primitives such as random oracles but block ciphers that perform encryption and decryption. The analysis presented in this paper focuses on adversaries that aim in producing outputs exhibiting patterns when attacking systems that output truly random data. Block ciphers, on the other hand, are permutations and do not output truly random data. In fact, it is well known that block ciphers can be distinguished from random oracles for sufficiently large query budgets (Hall, 1998), (Gilboa, 2015). In order for our analysis to be valid, we limit the adversary query budgets and their associated security levels to values for which block cipher outputs are practically indistinguishable from the truncated outputs of random oracles. For example, the 64-bit limit we mention above derives from the analysis found in references (Hall, 1998), (Gilboa, 2015).

4.1 Security of the Byte and Word Equality Patter Detectors

Byte and word equality checks are applicable to the protection of memory and storage data because many data units in computing systems contain code or data structures which demonstrate significant byte or word value replications. The entropy index E associated

with a data unit consisting of n bytes, which demonstrates m bytes being equal to each other is given by:

$$E \cong -\log_2 \left(\binom{n}{m} \left(\frac{1}{256}\right)^{m-1} \cdot \left(1 - \frac{1}{256}\right)^{n-m} \right) \quad (3)$$

Similarly the entropy index E associated with a data unit consisting of n 16-bit words, which demonstrates m words being equal to each other is given by:

$$E \cong -\log_2 \left(\binom{n}{m} \left(\frac{1}{65536}\right)^{m-1} \cdot \left(1 - \frac{1}{65536}\right)^{n-m} \right) \quad (4)$$

In order for an adversary to successfully attack the byte or the word equality pattern detector, the adversary needs to produce corrupted plaintext data demonstrating m or more equal byte/word values. We consider a simple adversary model, where the adversary repeatedly attempts to corrupt some ciphertext, so as to produce plaintext that demonstrates patterns. The only requirement we introduce for the adversary is that the number of adversary queries should be bounded so that the system which is being attacked (e.g., a wide block cipher) should be practically indistinguishable from a truncated output random oracle. The advantage of such adversary is equal to the probability of seeing some specific patterns among random data plus the advantage of distinguishing the real cryptographic system used (i.e., a pseudo-random permutation) from a truncated output random oracle. The security analysis, which will follow, will be focused on the probability of seeing specific patterns among random data.

For random data, byte or word equalities are observed with probability computed as the birthday collision probability associated n elements, m collisions and 256 or 65536 birthdays for bytes and words respectively. Such probability can be approximated in many ways, for example using a number of recent analytical results (Klamkin, 1967), (Suzuki, 2006), (Kounavis, 2017). Using the approximation from reference (Kounavis, 2017) we obtain:

$$P^{(bytes-equal)} \cong 1 - \prod_{i=0}^{n-1} \sum_{j=0}^{m-2} \binom{n-i-1}{j} \cdot \left(\frac{1}{256}\right)^{j} \cdot \left(\frac{255}{256}\right)^{n-i-j-1} \quad (5)$$

and

$$P^{(words-equal)} \cong 1 - \prod_{i=0}^{n-1} \sum_{j=0}^{m-2} \binom{n-i-1}{j} \cdot \left(\frac{1}{65536}\right)^{j} \cdot \left(\frac{65535}{65536}\right)^{n-i-j-1} \quad (6)$$

The advantage of an adversary who corrupts ciphertext values, hoping that his corruptions will pass undetected, $\mathbf{Adv}^{(C)}$, is computed from the probabilities $P^{(bytes\text{-}equal)}(n, m)$ and $P^{(words\text{-}equal)}(n, m)$ as follows:

$$\begin{aligned} &\mathbf{Adv}^{(C)} \leq P^{(bytes-equal)}(n,m) + \mathbf{Adv}^{(D)}, \\ &\text{for byte equalities} \\ &\mathbf{Adv}^{(C)} \leq P^{(words-equal)}(n,m) + \mathbf{Adv}^{(D)}, \\ &\text{for word equalities} \end{aligned} \quad (7)$$

where $\mathbf{Adv}^{(D)}$ is the advantage of distinguishing the cryptographic system used from a truncated output random oracle.

The birthday collision probabilities presented in equations 5 and 6 are a little higher than the probability values 2^{-E} associated with the entropy index values given above. This is because these birthday collision probabilities include all events where there are more than m values being equal in a data unit. In Tables 1 and 2, we show the cumulative entropy index distribution computed over 111 million cache lines for the byte and word equality pattern checks respectively. The corresponding threshold values associated with each entropy index value are also shown.

As is evident from the tables, entropy index values corresponding to reasonably high security levels (e.g., 32 bits, 24 bits) for implicit integrity are associated with high pass rates in regular client data cache lines. This is demonstrated by both the byte equality and the word equality pattern checks. For example, for the security level of 32 bits, the percentage of cache lines that demonstrate this entropy index or higher is 85.28% if the byte equality pattern check is used. For the security level of 24 bits, the percentage of cache lines that demonstrate this entropy index value or higher is increased to 91.48%.

Table 1: Entropy index distribution for the byte equality pattern check.

minimum entropy index (bits)	corresponding threshold	% of cache lines
8	5	97.27
16	7	93.55
24	8	91.48
32	10	85.28
40	11	83.30

Table 2: Entropy index distribution for the word equality pattern check.

minimum entropy index (bits)	corresponding threshold	% of cache lines
8	3	87.37
16	3	87.37
24	4	82.26
32	4	82.26
40	5	77.70

4.2 Security of a Detector That Combines All Pattern Checks

The entropy index and security levels associated with other types of pattern checks can be computed using similar analytical tools from probability theory. The pass rates for a scheme that combines all aforementioned pattern checks are shown in Table 3. For the 32-bit security level, the combined scheme demonstrates a pass rate of 91.11%. For the 24-bit security level the combined scheme demonstrates a pass rate of 94.48%.

When different pattern checks are combined, an attacker succeeds if the attacker produces any of the considered patterns over some corrupted plaintext. Due to this fact, the overall security level supported by the combined pattern detector is not equal to the maximum entropy index characterizing the patterns, but lower by some small parameter ϵ, which needs to be taken into account.

Table 3: Entropy index distribution for a scheme that combines all pattern checks of Figures 2 and 3.

minimum entropy index (bits)	patterns considered	% of cache lines
8	all (1-8)	98.13
16	all (1-8)	96.50
24	all (1-8)	94.48
32	all (1-8)	91.11
40	all (1-8)	88.88

Table 4: Synthesis results for the Extended Pattern Matching algorithm of Figure 2.

patterns	cycles	μm^2	cells
equal words	2	3523	15.8K
eq. words, adj. eq. bytes, bytes with spec. values, most sig. nibbles	3	7179	32.1K
eq. words, eq. bytes,	3	14350	64.2K
eq. words, eq. bytes, most sig. nibbles	3	17119	76.6K
all 8 patterns	4	26218	117.3K

4.3 Hardware Realization

Implementing the aforementioned pattern checks efficiently in hardware is feasible. This section is focused on the hardware implementation of the pattern detectors of Figure 2, as these are the most area and latency efficient. For the byte and word equality patterns, interconnect wiring is required, which connects every byte or word in a data unit with every other byte or word. The necessary amount XOR gates can be employed to perform comparisons between such entities, followed by a layer of AND gates and counters that compute the number of matches associated with each entity. The pattern detectors that compute the sum of the two highest nibble frequencies are the most complex. These circuits can be implemented using a combination of decoders, counters and trees of comparators. Each nibble value is first passed to a decoder circuit which generates 16 lines, each corresponding to a different nibble value from 0 to 15. The decoder outputs corresponding to the same nibble value are added to each other. There are 16 different summation outputs from this stage. These outputs constitute the histogram of nibble values produced from the input data. Our synthesis results produced using Intel's ® 14 nm process technology (Jan, 2015) and the frequency of 1.67 GHz are shown in Table 4.

5 DISCUSSION

One issue that needs addressing is how to detect corruption if data do not exhibit patterns. As we demonstrated above, patterns can be found in up to 91% of client workload cache lines and 84% of server workload cache lines. Whereas such data can be protected using implicit integrity, the remaining 9%-16% of the data need protecting also.

Such design issue can be easily addressed. Implementations can protect the overwhelming

majority of user data that exhibit patterns using implicit integrity and the remaining data using standard techniques. There is nothing in the implicit integrity methodology, discussed here, that prevents it from being used together with other independent integrity mechanisms such as MACs. Such solutions can co-exist with implicit integrity.

If some decrypted content exhibits patterns, then there is some assurance that no corruption has occurred. If no patterns are exhibited, however, a search can be made for a MAC associated with the content. If no MAC is found then the data is deemed corrupted. Otherwise, an integrity check is made using the returned MAC. Such implementation can use a content addressable memory unit or a hash table for accessing and managing MACs. Further investigation on hardware and operating system changes required in order to support implicit integrity are the subject of future work.

Finally, a reasonable question that can be asked is why not simply compress the data and augment it by a MAC in the now free space. Compressing and decompressing in combinatorial logic, for some patterns such as the nibble-based ones, can be in fact quite costly. Ongoing research of ours shows that the client cache lines that can be compressed at reasonable cost are significantly fewer than those protected via implicit integrity (78% as opposed to 91%). Such analysis is the subject of future work.

REFERENCES

D. Durham and M. Long, "Memory Integrity", *United States Patent*, No. 9,213,653, December 2013.

D. Durham, et. al., "Memory Integrity with Error Detection and Correction", *United States Patent*, No. 9,990,249, December 2015.

D. Durham, S. Chhabra, M. Kounavis, S. Deutsch, K. Grewal, J. Cihula and S. Komijani, "Convolutional Memory Integrity", *United States Patent Application*, No. 20170285976, 2016.

SHA256, "Secure Hash Standard", *Federal Information Processing Standards Publication* FIPS PUB 180-4, 2012.

SHA-3, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", *Federal Information Processing Standards Publication* FIPS PUB 202, 2015.

HMAC, "The Keyed-Hash Message Authentication Code (HMAC)", *Federal Information Processing Standards Publication* FIPS PUB 198-1, 2008.

KMAC, "SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash", *NIST Special Publication* 800-185, 2016.

AES, "Advanced Encryption Standard (AES)", *Federal Information Processing Standards Publication* FIPS PUB 197, 2001.

N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Calas and J. Walker, "The Skein Hash Function Family", *available online* at http://www.skein-hash.info/sites/default/files/skein1.2.pdf, 2009.

F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue and U. Savagaonkar, "Innovative instructions and software model for isolated execution", *Proceedings of the Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

C. Hall, D. A. Wagner, J. Kelsey and B. Schneier, "Building PRFs from PRPs", *CRYPTO 1998*: 370-389.

S. Gilboa and S. Gueron, "Distinguishing a truncated random permutation from a random function", *IACR Cryptology ePrint Archive* 2015: 773 (2015).

M. S. Klamkin and D. J. Newman, "Extensions on the Birthday Surprise" *Journal of Combinatorial Theory*, Vol. 3, pp. 279-282, 1967.

K. Suzuki, D. Tonien, K. Kurosawa and K. Toyota, "Birthday Paradox for Multi-collisions", *International Conference on Information Security and Cryptology*, pp. 29-40, 2006.

M. Kounavis, S. Deutsch, D. Durham and S. Komijani, "Non-recursive computation of the probability of more than two people having the same birthday", *ISCC 2017*: 1263-1270.

C. Jan et al., "A 14 nm SoC Platform Technology Featuring 2nd Generation Tri-Gate Transistors, 70 nm Gate Pitch, 52 nm Metal Pitch, and 0.0499 μm^2 SRAM Cells, Optimized for Low Power, High Performance and High Density SoC Products", pp. 12-13, *Sym. on VLSI Tech*., 2015.

V. T. Hoang, T. Krovetz and P. Rogaway, "Robust Authenticated-Encryption AEZ and the Problem That It Solves", *EUROCRYPT*, 2015.