Incremental Multiset Hash Functions and Their Application to Memory Integrity Checking

Dwaine Clarke*, Srinivas Devadas, Marten van Dijk**, Blaise Gassend, G. Edward Suh

MIT Computer Science and Artificial Intelligence Laboratory
{declarke, devadas, marten, gassend, suh}@mit.edu

Abstract. We introduce a new cryptographic tool: *multiset* hash functions. Unlike standard hash functions which take strings as input, multiset hash functions operate on multisets (or sets). They map multisets of arbitrary finite size to strings (hashes) of fixed length. They are incremental in that, when new members are added to the multiset, the hash can be updated in time proportional to the change. The functions may be *multiset-collision resistant* in that it is difficult to find two multisets which produce the same hash, or just *set-collision resistant* in that it is difficult to find a set and a multiset which produce the same hash.

We demonstrate how set-collision resistant multiset hash functions make an existing offline memory integrity checker secure against active adversaries. We improve on this checker such that it can use smaller time stamps without increasing the frequency of checks. The improved checker uses multiset-collision resistant multiset hash functions.

Keywords: multiset hash functions, set-collision resistance, multiset-collision resistance, incremental cryptography, memory integrity checking

1 Introduction

Standard hash functions, such as SHA-1 [11] and MD5 [12], map strings of arbitrary finite length to strings (hashes) of a fixed length. They are collision-resistant in that it is difficult to find different input strings which produce the same hash. Incremental hash functions, described in [2], have the additional property that, given changes to the input string, the computation to update the hashes is proportional to the amount of change in the input string. For a small change, incremental hashes can be quickly updated, and do not need to be recalculated over the entire new input.

Multiset hash functions are a novel cryptographic tool, for which the ordering of the inputs is not important. They map multisets of arbitrary finite size to hashes of fixed length. They are incremental in that, when new members

* Note: authors are listed alphabetically.

** Visiting researcher from Philips Research, Prof Holstlaan 4, Eindhoven, The Netherlands.

are added to the multiset, the hash can be quickly updated. Because multiset hash functions work on multisets, we introduce definitions for multiset-collision resistance and set-collision resistance.

In particular, we introduce four multiset hash functions, each with its own advantages. `MSet-XOR-Hash` uses the XOR operation and is very efficient; however, it uses a secret key and is only set-collision resistant. `MSet-Add-Hash` uses addition modulo a large integer and, thus, is slightly less efficient than `MSet-XOR-Hash`; `MSet-Add-Hash` also uses a secret key but it is multiset-collision resistant. `MSet-Mu-Hash` uses finite field arithmetic and is not as efficient as the other two hash functions; however, `MSet-Mu-Hash` is multiset-collision resistant, and unlike the other two hash functions, does not require a secret key. `MSet-VAdd-Hash` is more efficient than `MSet-Mu-Hash`; it is also multiset-collision resistant, and does not use a secret key, but the hashes it produces are significantly longer than the hashes of the other functions.

The proven security of `MSet-XOR-Hash` and `MSet-Add-Hash` is quantitative. We reduce the hardness of finding collisions to the hardness of breaking the underlying pseudorandom functions. The proven security of `MSet-Mu-Hash` is in the random oracle model and is based on the hardness of the discrete logarithm problem. The proven security of `MSet-VAdd-Hash` is also in the random oracle model and is based on the hardness of the worst-case shortest vector problem.

We demonstrate how multiset hash functions enable secure offline integrity checkers for untrusted memory. Checking the integrity of memory is important in building secure processors which can facilitate software licensing and Digital Rights Management (DRM) [13, 14].

The paper is organized as follows. Section 2 describes related work and summarizes our contributions. Multiset hash functions are defined in Section 3. `MSet-XOR-Hash` and `MSet-Add-Hash` are described in Section 4; `MSet-Mu-Hash` and `MSet-VAdd-Hash` are described in Section 5. Our application of multiset hash functions to checking the integrity of memory is detailed in Section 6. Section 7 concludes the paper. Appendices A, B, C, and D prove the security of our multiset hash functions. Appendix E proves the security of our memory integrity checker.

2 Related Work and Our Contributions

The main contribution of our work is the introduction of multiset hash functions together with the definition of multiset and set collision resistance. The second contribution is the development of a general theory leading to Theorem 1 from which we derive set-collision resistance for `MSet-XOR-Hash`, a multiset hash based on the XOR operation (addition modulo 2), and multiset-collision resistance for `MSet-Add-Hash`, a multiset hash based on addition modulo a large integer. The theory generalizes the results in [3], where an XOR-based scheme is used for message authentication. Our theory holds for addition modulo any integer.

Both `MSet-XOR-Hash` and `MSet-Add-Hash` use a secret key. The third contribution is Theorem 2 that proves multiset-collision resistance for `MSet-Mu-Hash`, a multiset hash function based on multiplication in a finite field; `MSet-Mu-Hash` does not use a secret key. The proof's basic line of thought is from [4] which develops message hashing based on multiplication in a finite field. The fourth contribution, leading to `MSet-VAdd-Hash`, is Theorem 3 proving that we may replace multiplication in the finite field by vector addition modulo a large integer. In [4], a similar theorem is used for message hashing. Our theorem (and their theorem) follows directly from application of Ajtai's theorem [1, 8].

Our final significant contribution is that we introduce an offline checker that is cryptographically secure against active adversaries, and which improves on the performance of the original offline checker in [6].

3 Multiset Hash Functions

This section describes multiset hash functions. We first introduce multisets. We refer to a multiset as a finite unordered group of elements where an element can occur as a member more than once. All sets are multisets, but a multiset is not a set if an element appears more than once. Let M be a multiset of elements of a countable set B. The number of times $b \in B$ is in the multiset M is denoted by M_b and is called the multiplicity of b in M. The sum of all the multiplicities of M is called the cardinality of M. Multiset union combines two multisets into a multiset in which elements appear with a multiplicity that is the sum of their multiplicities in the initial multisets. We denote multiset union by $\cup$ and assume that the context in which $\cup$ is used makes clear to the reader whether we mean set union or multiset union.

Definition 1. *Let $(\mathcal{H}, +_{\mathcal{H}}, \equiv_{\mathcal{H}})$ be a triple of probabilistic polynomial time (ppt) algorithms. That triple is a* multiset hash function *if it satisfies:*

compression: *$\mathcal{H}$ maps multisets of B into elements of a set with cardinality $\approx 2^m$, where m is some integer. Compression guarantees that we can store hashes in a small bounded amount of memory.*

comparability: *Since $\mathcal{H}$ can be a probabilistic algorithm, a multiset need not always hash to the same value. Therefore we need $\equiv_{\mathcal{H}}$ to compare hashes. The following relation must hold for comparison to be possible:*

$$\mathcal{H}(M) \equiv_{\mathcal{H}} \mathcal{H}(M)$$

for all multisets M of B.

incrementality: *We would like to be able to efficiently compute $\mathcal{H}(M \cup M')$ knowing $\mathcal{H}(M)$ and $\mathcal{H}(M')$. The $+_{\mathcal{H}}$ operator makes that possible:*

$$\mathcal{H}(M \cup M') \equiv_{\mathcal{H}} \mathcal{H}(M) +_{\mathcal{H}} \mathcal{H}(M')$$

for all multisets M and M' of B. In particular, knowing only $\mathcal{H}(M)$ and an element $b \in B$, we can easily compute $\mathcal{H}(M \cup \{b\}) = \mathcal{H}(M) +_{\mathcal{H}} \mathcal{H}(\{b\})$.

As it is, this definition is not very useful, because $\mathcal{H}$ could be any constant function. We need to add some kind of collision resistance to have a useful hash function. A collision for M' is a multiset $M \neq M'$ such that $\mathcal{H}(M) \equiv_{\mathcal{H}} \mathcal{H}(M')$. A multiset hash function is *(multi)set-collision resistant* if it is computationally infeasible to find a (multi)set S of B and a multiset M of B such that the cardinalities of S and M are of polynomial size in m, $S \neq M$, and $\mathcal{H}(S) \equiv_{\mathcal{H}} \mathcal{H}(M)$. The following definition makes this notion formal.

Definition 2. *Let a family $\mathcal{F}$ of multiset hash functions $(\mathcal{H}_K, +_{\mathcal{H}_K}, \equiv_{\mathcal{H}_K})$ be indexed by a key (seed) $K \in \mathcal{K}$. For $\mathcal{H}_K$ in $\mathcal{F}$, we denote by m_K the logarithm of the cardinality of the set into which $\mathcal{H}_K$ maps multisets of B, that is m_K is the number of output bits of $\mathcal{H}_K$. We define $\mathcal{K}_m$ as the set of keys $K \in \mathcal{K}$ for which $m_K \geq m$. By $\mathcal{A}(\mathcal{H}_K)$ we denote a probabilistic polynomial time (in m_K) algorithm with oracle access to $(\mathcal{H}_K, +_{\mathcal{H}_K}, \equiv_{\mathcal{H}_K})$.*

The family $\mathcal{F}$ satisfies (multi)set-collision resistance if for all ppt algorithms $\mathcal{A}(.)$, any number c, and m large enough (with respect to c),[1]

$$Prob \left\{ \begin{array}{l} K \leftarrow \mathcal{K}_m, (S, M) \leftarrow \mathcal{A}(\mathcal{H}_K) : \\ S \textit{ is a (multi)set and } M \textit{ is a multiset of } B \\ \ \textit{such that } S \neq M \textit{ and } \mathcal{H}_K(S) \equiv_{\mathcal{H}_K} \mathcal{H}_K(M) \end{array} \right\} < m^{-c}.$$

Note that because $\mathcal{A}(\mathcal{H}_K)$ is polynomial in m_K, we will consider that it can only output polynomial sized S and M. We are disallowing compact representations for multisets that would allow $\mathcal{A}(.)$ to express larger multisets (such compact representations do not lead to a feasible attack in our offline memory integrity application).

4 Additive Multiset Hash

In this section we give an example of a construction of (multi)set-collision resistant multiset hash functions. Let $B = \{0, 1\}^m$ represent the set of bit vectors of length m and let M be a multiset of elements of B. Recall that the number of times $b \in B$ is in the multiset M is denoted by M_b and is called the multiplicity of b in M. Let $H_K : \{0, 1\}^{m+1} \rightarrow \mathbb{Z}_n^l$ be randomly selected from a pseudorandom family of hash functions [9]. Let

$$L \approx n^l \approx 2^m, L \leq n^l, L \leq 2^m,$$

and define

$$\mathcal{H}_K(M) = \left[H_K(0, r) + \sum_{b \in B} M_b H_K(1, b) \mod n \ ; \ \sum_{b \in B} M_b \mod L \ ; \ r \right] \Bigg|_{r \leftarrow B},$$

[1] The probability is taken over a random selection of K in $\mathcal{K}_m$ (denoted by $K \leftarrow \mathcal{K}_m$) and over the randomness used in the ppt algorithm $\mathcal{A}(\mathcal{H}_K)$ (denoted by $(S, M) \leftarrow \mathcal{A}(\mathcal{H}_K)$).

where $r \in B$ is a random nonce[2]. Notice that the logarithm of the cardinality m_K of the set into which $\mathcal{H}_K$ maps multisets of B is equal to

$$m_K = \log(n^l) + \log(L) + \log(2^m) \approx 3m.$$

We say two triples $[h, c, r]$ and $[h', c', r']$ are equivalent, $[h; c; r] \equiv_{\mathcal{H}_K} [h'; c'; r']$, if and only if $h - H_K(0, r) = h' - H_K(0, r')$ modulo n and $c = c'$ modulo L. Notice that checking whether $\mathcal{H}_K(M) \equiv_{\mathcal{H}_K} \mathcal{H}_K(M')$ is efficient. We define addition of two triples $[h; c; r] +_{\mathcal{H}_K} [h'; c'; r']$ by the result of the computation

$$[H_K(0, r'') + h - H_K(0, r) + h' - H_K(0, r') \mod n \ ; \ c + c' \mod L \ ; \ r'']|_{r'' \leftarrow B}.$$

Clearly, $\mathcal{H}_K(M \cup M') \equiv_{\mathcal{H}_K} \mathcal{H}_K(M) +_{\mathcal{H}_K} \mathcal{H}_K(M')$, hence, $(\mathcal{H}_K, +_{\mathcal{H}_K}, \equiv_{\mathcal{H}_K})$ is a multiset hash. The proof of the next theorem is in Appendix A.

Theorem 1. *It is computationally infeasible to find a multiset M with multiplicities $< n$ and a multiset M' such that the cardinalities of M and M' are polynomial sized in m, $M \neq M'$, and $\mathcal{H}_K(M) \equiv_{\mathcal{H}_K} \mathcal{H}_K(M')$.*

As an example we consider $n = 2$ and $l = m$. Then the condition that a multiset M has multiplicities < 2 simply means that M is a set. This leads to set-collision resistance. Furthermore notice that addition modulo 2 defines xor $\oplus$.

Corollary 1. *(`MSet-XOR-Hash`) The multiset hash corresponding to*

$$\mathcal{H}_K(M) = \left[H_K(0, r) \oplus \bigoplus_{b \in B} M_b H_K(1, b) \ ; \ \sum_{b \in B} M_b \mod 2^m \ ; \ r \right] \Bigg|_{r \leftarrow B},$$

where $H_K : \{0, 1\} \times B \to \mathbb{Z}_2^m$ is randomly selected from a pseudorandom family of hash functions, is set-collision resistant.

Notice that $\mathcal{H}_K(M)$ keeps track of the cardinality of M. If this were not the case then $\mathcal{H}_K(S)$ and $\mathcal{H}_K(M)$ are equivalent for any S and M with $S_b = M_b$ modulo $n = 2$ for $b \in B$. This would contradict set-collision resistance. Also notice that $r \leftarrow B$ is randomly chosen. If r was a fixed known constant, then knowledge of n tuples $[M^i \ ; \ \mathcal{H}_K(M^i)]$ reveals n vectors

$$\bigoplus_{b \in B} M_b^i H_K(1, b) \in \mathbb{Z}_2^m.$$

If $n = 2m$ then with high probability these n vectors span the vector space $\mathbb{Z}_2^m$. This means that each vector in $\mathbb{Z}_2^m$ can be constructed as a linear combination of these n vectors [4]:

$$\bigoplus_{i=1}^{n} a_i \cdot \left(\bigoplus_{b \in B} M_b^i H_K(1, b) \right) = \bigoplus_{b \in B} \left(\bigoplus_{i=1}^{n} a_i M_b^i \right) H_K(1, b).$$

[2] Note, the set from which r is taken could be smaller than B.

Hence, a polynomial sized collision can be constructed for any polynomial sized M.

In Appendix B we show that for n exponentially large in m, we may remove the cardinality $\sum_{b\in B} M_b$ from the scheme altogether. By taking $l = 1$ and $L = n = 2^m$ we obtain the next corollary.

Corollary 2. *(`MSet-Add-Hash`) The multiset hash corresponding to*

$$\mathcal{H}_K(M) = \left[H_K(0,r) + \sum_{b\in B} M_b H_K(1,b) \mod 2^m \ ; \quad r \right]\Bigg|_{r\leftarrow B},$$

where $H_K : \{0,1\} \times B \to \mathbb{Z}_{2^m}$ is randomly selected from a pseudorandom family of hash functions, is multiset collision resistant.

The main difference between the `MSet-XOR-Hash` and `MSet-Add-Hash` is binary addition without and with carry respectively. This leads to either set collision resistance or multiset collision resistance.

In Appendix B we show that it is possible to replace the random nonce r by a counter that gets incremented on each use of $\mathcal{H}_K$. This removes the need for a random number generator from the scheme. Moreover, shorter values can be used for r as long as the key is changed when r overflows; this reduces the size of the hash. Also if the weighted sum of the hashes $H_K(1,b)$ in $\mathcal{H}_K(M)$ is never revealed to the adversary then we can remove $H_K(0,r)$ from the scheme altogether. For example, in the case where the weighted sums are encrypted by using a pseudorandom family of permutations (see Corollary 4 in Appendix B).

5 Multiplicative Multiset Hash

A multiset-collision resistant multiplicative multiset hash can be defined as follows. Let q be a large prime power and consider the computations in the field $GF(q)$. Let $H : B \to GF(q)$ be a poly-random function [9], that is, no polynomial time (in the logarithm of q) algorithm with oracle access H can distinguish between values of H and true random strings, even when the algorithm is permitted to select the arguments to H (in practice one would use MD5 [12] or SHA1 [11]). We define

$$\mathcal{H}(M) = \prod_{b\in B} H(b)^{M_b}, \tag{1}$$

$\equiv_{\mathcal{H}}$ to be equal to $=$, and $+_{\mathcal{H}}$ to be multiplication in $GF(q)$.

Clearly, $(\mathcal{H}, +_{\mathcal{H}}, \equiv_{\mathcal{H}})$ is a multiset hash. An advantage of the scheme is that we do not need a secret key. Unfortunately it relies on finite field arithmetic, which makes it too costly for some applications.

The proof of the following theorem is given in Appendix C, where we also define the discrete log (DL) assumption which says that for random $y \in GF(q)$ and generator $g \in GF(q)$, it is computationally infeasible to find x such that $g^x = y$ (x is called the discrete log of y).

Theorem 2. *(`MSet-Mu-Hash`) Under the DL assumption, the family*[3] *of multiset hash functions,* $(\mathcal{H}, +_{\mathcal{H}}, \equiv_{\mathcal{H}})$*, as defined in (1), is multiset collision resistant.*

Under certain assumptions we may replace multiplication in $GF(q)$ by addition modulo a large number. Even though the number of output bits of the resulting multiset hash needs to be much larger (since it is based on 'weaker' assumptions), the overall solution becomes more efficient since no finite field arithmetic is needed. Let $H : B \rightarrow \mathbb{Z}_n^l$, $n = 2^{\sqrt{m}}$, $l = \sqrt{m}$, be a poly-random function. Now, we define

$$\mathcal{H}(M) = \sum_{b \in B} M_b H(b) \mod n, \tag{2}$$

$\equiv_{\mathcal{H}}$ to be equal to $=$, and $+_{\mathcal{H}}$ to be vector addition modulo n. See Appendix D for the proof of the next theorem and the definition of the worst-case shortest vector (SV) problem.

Theorem 3. *(`MSet-VAdd-Hash`) By assuming that the SV problem is infeasible to solve in polynomial time, the family*[4] *of multiset hash functions,* $(\mathcal{H}, +_{\mathcal{H}}, \equiv_{\mathcal{H}})$*, as defined in (2), is multiset collision resistant.*

Remark. Because H can be evaluated with oracle access to $\mathcal{H}$, Theorems 2 and 3 still hold for a stronger form of multiset-collision resistance, in which it is computationally infeasible for an adversary with oracle access to H (instead of $\mathcal{H}$) to find a collision. This is what allows to use a publicly available H.

6 Integrity Checking of Random Access Memory

We now show how our multiset hash functions can be used to build secure offline integrity checkers for memory. Section 6.1 explains the model, and Section 6.2 shows our offline checker. Our implementation of this checker in the AEGIS secure processor [13] is described in [14, 7].

6.1 Model

Figure 1 illustrates the model we use. There is a checker that keeps and maintains some small, fixed-sized, trusted state. The untrusted RAM (main memory) is arbitrarily large. The finite state machine (FSM) generates loads and stores and the checker updates its trusted state on each FSM load or store to the untrusted RAM. The checker uses its trusted state to verify the integrity of the untrusted RAM. The trusted computing base (TCB) consists of the FSM, and the checker with its trusted state. For example, the FSM could be a processor. The checker would be special hardware that is added to the processor to detect tampering in the external memory.

---

[3] The family is seeded by $GF(q)$.

[4] The family is seeded by $\mathbb{Z}_n^l$.

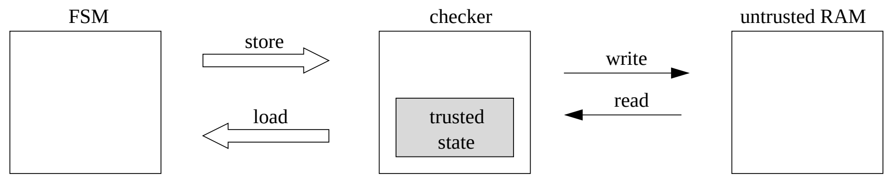

Fig. 1. Model

The checker checks if the untrusted RAM behaves correctly, i.e. like valid RAM. *RAM behaves like valid RAM if the data value that the checker reads from a particular address is the same data value that the checker had most recently written to that address.* In our model, the untrusted RAM is assumed to be actively controlled by an adversary. The untrusted RAM may not behave like valid RAM if the RAM has malfunctioned because of errors, or if it has been somehow altered by the adversary.

For this problem, a simple solution such as calculating a message authentication code (MAC) of the data value and address, writing the (data value, MAC) pair to the address, and using the MAC to check the data value on each read, does not work. The approach does not prevent replay attacks: an adversary can replace the (data value, MAC) pair currently at an address with a different pair that was previously written to the address. The essence of an offline checker is that a "log" of the sequence of FSM operations is maintained in fixed-sized trusted state in the checker.

6.2 Offline Checker

Figure 2 shows the basic `put` and `get` operations that are used internally in the checker. Figure 3 shows the interface the FSM calls to use the offline checker to check the integrity of the memory.

In Figure 2, the checker maintains two multiset hashes and a counter. In memory, each data value is accompanied by a time stamp. Each time the checker performs a `put` operation, it appends the current value of the counter (a time stamp) to the data value, and writes the (data value, time stamp) pair to memory. When the checker performs a `get` operation, it reads the pair stored at an address, and, if necessary, updates the counter so that it is strictly greater than the time stamp that was read. The multiset hashes are updated ($+_{\mathcal{H}}$) with (a, v, t) triples corresponding to the pairs written or read from memory.

Figure 3 shows how the checker implements the `store-load` interface. To initialize the RAM, the checker `puts` an initial value to each address. When the FSM performs a `store` operation, the checker `gets` the original value at the address, then `puts` the new value to the address. When the FSM performs a `load` operation, the checker `gets` the original value at the address and returns this value to the FSM; it then `puts` the same value back to the address. To check the integrity of the RAM at the end of a sequence of FSM stores and

The checker's fixed-sized state is:

- 2 multiset hashes: WriteHash and ReadHash. Initially both hashes are 0.
- 1 counter: Timer. Initially Timer is 0.

`put`(a, v) writes a value v to address a in memory:

1. Let t be the current value of Timer. Write (v, t) to a in memory.
2. Update WriteHash: WriteHash $+_{\mathcal{H}}=$ hash(a, v, t).

`get`(a) reads the value at address a in memory:

1. Read (v, t) from a in memory.
2. Update ReadHash: ReadHash $+_{\mathcal{H}}=$ hash(a, v, t).
3. Timer = max(Timer, $t + 1$).

Fig. 2. `put` and `get` operations

loads, the checker `gets` the value at each address, then compares WriteHash and ReadHash. If WriteHash is equal to ReadHash, the checker concludes that the RAM has been behaving correctly.

Because the checker checks that WriteHash is equal to ReadHash, substitution (the RAM returns a value that is never written to it) and replay (the RAM returns a stale value instead of the one that is most recently written) attacks on the RAM are prevented. The purpose of the time stamps is to prevent reordering attacks in which RAM returns a value that has not yet been written so that it can subsequently return stale data. Suppose we consider the `put` and `get` operations that occur on a particular address as occurring on a timeline. Line 3 in the `get` operation ensures that, for each `store` and `load` operation, each write has a time stamp that is strictly greater than all of the time stamps previously read from memory. Therefore, the first time an adversary tampers with a particular (data value, time stamp) pair that is read from memory, there will not be an entry in the WriteHash matching the adversary's entry in the ReadHash, and that entry will not be added to the WriteHash at a later time.

The Timer is not solely under the control of the checker, and is a function of what is read from memory, which is untrusted. Therefore, the WriteHash cannot be guaranteed to be over a set. For example, for a sequence of store and load operations occurring on the same address, an adversary can decrease the time stamp that is stored in memory and have triples be added to the WriteHash multiple times. The ReadHash can also not be guaranteed to be over a set because the adversary controls the pairs that are read from memory. Thus, set-collision resistance is not sufficient, and we require multiset-collision resistant hash functions.

The proof of the following theorem is in Appendix E.

`initialize`() initializes RAM.

1. `put`$(a, 0)$ for each address a.

`store`(a, v) stores v at address a.

1. `get`(a).
2. `put`(a, v).

`load`(a) loads the data value at address a.

1. $v =$ `get`(a). Return v to the caller.
2. `put`(a, v).

`check`() checks if the RAM has behaved correctly (at the end of operation).

1. `get`(a) for each address a.
2. If WriteHash is equal to ReadHash, return true.

Fig. 3. Offline integrity checking of random access memory

Theorem 4. *Let W be the multiset of triples written to memory and let R be the multiset of triples read from memory. That is, W hashes to* WriteHash *and R hashes to* ReadHash*. Suppose the accesses to each address are an alternation of* `puts` *and* `gets`*. If the RAM does not behave like valid RAM, then $W \neq R$.*

The following corollary shows the hardness of breaking our offline memory integrity checking scheme.

Corollary 3. *Tampering with the RAM without being detected is as hard as finding a collision $W \neq R$ for the multiset hash function.*

Offline memory integrity checking was introduced by Blum et al. [6]. However, the original offline checker in [6] differs from our checker in two respects. First, the original checker is implemented with ϵ-biased hash functions [10]. These hash functions are set-collision resistant against random errors but not against a malicious adversary. Secondly, the Timer is incremented on each `put` operation and is not a function of what is read from memory. The Timer is solely under the control of the checker. This means that the pairs that are used to update WriteHash form a set. Therefore set-collision resistance is sufficient. The original offline checker can be made secure against active adversaries by using a set-collision resistant multiset hash function, instead of ϵ-biased hash functions. Our offline checker improves on the original checker because Timer is not incremented on every `load` and `store` operation. Thus, time stamps can be smaller without increasing the frequency of checks, which improves the performance of the checker.

7 Conclusion

We have introduced incremental multiset hash functions which can be efficiently updated, and for which the ordering of inputs is not important. Table 1 summarizes our comparison of the multiset hash functions introduced in this paper. In the table, we indicate whether the security is based on pseudorandom family of hash functions (PRF), the random oracle model (RO), the discrete log assumption (DL), or/and the hardness of the worst case shortest vector problem (SV). If hashes are to be visible to the adversary (i.e., the adversary can see the hashes in the trusted state, but cannot modify them), we indicate whether a random nonce/counter (r), or encryption is necessary. We have improved the security and the performance of the offline memory integrity checker in [6] as one application of these functions.

Table 1. Comparison of the Multiset Hash Functions

	collision resistance	key	security based on	comput. efficiency	length of output	offline checker	hash visible
`MSet-XOR-Hash`	set	Y	PRF	++	+	original	r/enc
`MSet-Add-Hash`	multiset	Y	PRF	++	+	both	r/enc
`MSet-Mu-Hash`	multiset	N	RO/DL	−	+	both	
`MSet-VAdd-Hash`	multiset	N	RO/SV	+	−	both	

References

1. M. Ajtai. Generating hard instances of lattice problems. In *28th ACM STOC*, pages 99–108, 1996.
2. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *Crypto '94*, LNCS 839. Springer-Verlag, 1994.
3. M. Bellare, R. Guerin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *Crypto '95*, LNCS 963. Springer-Verlag, 1995.
4. M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Eurocrypt '97*, LNCS 1233. Springer-Verlag, 1997.
5. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS '93*, pages 62–73. ACM Press, 1993.
6. M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. In *Algorithmica*, volume 12, pages 225–244, 1994.
7. D. Clarke, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. Offline integrity checking of untrusted storage. In *MIT-LCS-TR-871*, Nov. 2002.
8. O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems. In *Theory of Cryptography Library 96-09*, July 1996.
9. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):210–217, 1986.

10. J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. In *22nd ACM STOC*, pages 213–223, 1990.
11. NIST. FIPS PUB 180-1: Secure Hash Standard, April 1995.
12. R. Rivest. RFC 1321: The MD5 Message-Digest Algorithm, April 1992.
13. G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the* 17^{th} *Int'l Conference on Supercomputing*, June 2003.
14. G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Efficient memory integrity verification and encryption for secure processors. In *Proceedings of the* 36^{th} *Int'l Symposium on Microarchitecture*, Dec 2003.

A Proof of Collision Resistance of Additive Hash

Let $\mathcal{G}_m$ be the family of matrices with 2^{m+1} rows, l columns, and entries in $\mathbb{Z}_n$ (recall $L \approx n^l \approx 2^m$). Let H_K be a random matrix in $\mathcal{G}_m = \{H_1, H_2, H_3, \ldots\}$. Notice that H_K is the K-th matrix in $\mathcal{G}_m$. We assume that this matrix, or equivalently its label K, is secret and only accessible by the secure processor. The family of matrices $\mathcal{G}_m$ from which H_K is selected is publicly known.

The rows of H_K are labelled by $x \in \{0,1\}^{m+1}$ and denoted by $H_K(x)$. This represents H_K as a function from $x \in \{0,1\}^{m+1}$ to $\mathbb{Z}_n^l$, the set of vectors with length l and entries in $\mathbb{Z}_n$. In practice, H_K is not a completely random matrix over $\mathbb{Z}_n$, but H_K is selected from a pseudorandom family of functions. We address this issue as soon as we are ready to formulate a proof of Theorem 1.

The following theorem is about the probability that an adversary finds a collision for some multiset M'. The probability is taken over random matrices H_K in $\mathcal{G}_m$ ($H_K \leftarrow \mathcal{G}_m$) and the randomness of the random nonce used in $\mathcal{H}_K$.

Theorem 5. *Let M and M' be multisets of B. Let d be the greatest common divisor*[5] *of n and each of the differences $|M_b - M'_b|$, $b \in B$. Given knowledge of u tuples $[M^i\ ;\ \mathcal{H}_K(M^i)]$, the probability that M is a collision for M' is at most $u^2/2^m + (d/n)^l$.*

We first introduce some notation. Let $v(r, M)$ be the vector of length 2^{m+1} defined by

$$v(r, M)_{(0,b)} = 1 \text{ if and only if } b = r$$

and

$$v(r, M)_{(1,b)} = M_b.$$

Let $v(M)$ be the vector of length 2^{m+1} defined by $v(M)_{(0,b)} = 0$ and $v(M)_{(1,b)} = M_b$.

Lemma 1. *(i) Knowing $[M\ ;\ \mathcal{H}_K(M)]$ is equivalent to knowing*

$$[v(r, M)\ ;\ v(r, M)H_K \mod n].$$

(ii) $\mathcal{H}_K(M) \equiv_{\mathcal{H}_K} \mathcal{H}_K(M')$ if and only if $v(M)H_K = v(M')H_K$ modulo n and $\sum_{b \in B} M_b = \sum_{b \in B} M'_b$ modulo L.

[5] The greatest common divisor of 0 with a positive integer i is equal to i.

Proof. Notice that $v(r, M)$ encodes r, M, and, hence, the cardinality $\sum_{b \in B} M_b$ of M, and notice that

$$\mathcal{H}_K(M) = \left[v(r, M)H_K \mod n \; ; \; \sum_{b \in B} M_b \mod L \; ; \; r \right].$$

The lemma follows immediately from these observations.

Suppose that an adversary learns u tuples $[M^i \; ; \; \mathcal{H}_K(M^i)]$ or, according to Lemma 1.(i), u vectors $v(r^i, M^i)$ together with the corresponding $v(r^i, M^i)H_K$ modulo n. Let A be the $u \times 2^{m+1}$ matrix with rows $v(r^i, M^i)$. Then the matrix with rows $v(r^i, M^i)H_K$ is equal to AH_K. Clearly, A modulo n has full rank over $\mathbb{Z}_n$ if all r^i are different. The probability that there are two equal r^i's is at most $u^2/2^m$.

Lemma 2. *The probability that the r^i's corresponding to matrix A are all different is at least $1 - u^2/2^m$.*

By Lemma 1.(ii), in order to find a collision for M', the adversary needs to find a multiset $M \neq M'$ such that $v(M)H_K = v(M')H_K$ modulo n and such that the cardinalities of M and M' are equal to one another modulo L. The next three lemmas show how difficult this is for the adversary if he is in the situation of the previous lemma.

Lemma 3. *Let M and M' be multisets of B. The probability that $v(M)H_K = v(M')H_K$ modulo n is statistically independent of the knowledge of a full rank matrix A over $\mathbb{Z}_n$ corresponding to different r^i's and the knowledge of $h = AH_K$ modulo n.*

Proof. W.l.o.g. (after reordering the columns of A and the corresponding entries of $v(M) - v(M')$ and corresponding rows of H_K) matrix A has the form $A = (I \; A^1)$, where I is the $u \times u$ identity matrix, and $v(M) - v(M')$ has the form $(0 \; v)$, where 0 is the all zero vector of length u. Denote the top u rows of H_K by H_K^0 and let H_K^1 be such that

$$H_K = \begin{pmatrix} H_K^0 \\ H_K^1 \end{pmatrix}.$$

Clearly, the equation $h = AH_K$ modulo n is equivalent to

$$h = H_K^0 + A^1 H_K^1 \mod n. \tag{3}$$

The equation $0 = (v(M) - v(M'))H_K$ modulo n is equivalent to

$$0 = vH_K^1 \mod n. \tag{4}$$

Straightforward counting tells us that $Prob\{(4)|(3)\}$ is equal to the # of matrices H_K^1 satisfying (4) divided by the total # of matrices H_K^1. This is in turn equal to the # of matrices H_K satisfying (4) divided by the total # of matrices H_K, which is $Prob\{(4)\}$.

Lemma 4. *Let M and M' be multisets of B. Let d be the greatest common divisor of n and each of the differences $|M_b - M'_b|$, $b \in B$. Then $(v(M) - v(M'))H_K$ modulo n is uniformly distributed in $d\mathbb{Z}_n^l$.*

Proof. To prove this lemma, we show that each entry of $(v(M) - v(M'))H_K$ modulo n is uniformly distributed in $d\mathbb{Z}_n$. Let y represent one of the columns of H_K and define for $\beta \in \mathbb{Z}_n$ the set

$$\mathcal{C}_\beta = \{y : (v(M) - v(M'))y = \beta \mod n\}.$$

Since d divides each entry of $v(M) - v(M')$, it also divides the product $(v(M) - v(M'))y$, hence, $\mathcal{C}_\beta = \emptyset$ if β is not divisible by d. Since d is the greatest common divisor of n and each of the entries of $v(M) - v(M')$, there exists a vector y such that $(v(M) - v(M'))y = d$ modulo n. This proves that $\mathcal{C}_\beta \neq \emptyset$ if and only if d divides β. For a fixed column $y' \in \mathcal{C}_\beta \neq \emptyset$, the mapping $y \in \mathcal{C}_\beta \to y - y' \in \mathcal{C}_0$ is a bijection. Hence, the non-empty sets $\mathcal{C}_\beta$ have equal cardinality. We conclude that each entry of $(v(M) - v(M'))H_K$ modulo n is uniformly distributed in $d\mathbb{Z}_n$.

Lemma 5. *Let M and M' be multisets of B. Let d be the greatest common divisor of n and each of the differences $|M_b - M'_b|$, $b \in B$. Given knowledge of a full rank matrix A over $\mathbb{Z}_n$ corresponding to different r^i's and given knowledge of $h = AH_K$ modulo n, the probability that $v(M)H_K = v(M')H_K$ modulo n is equal to $(d/n)^l$.*

Proof. By Lemma 3, since matrix A corresponds to different r^i's and $(v(M) - v(M'))_{(0,r^i)} = 0$, the probability that the randomly chosen matrix H_K satisfies $0 = (v(M) - v(M'))H_K$ modulo n is independent of the knowledge of $h = AH_k$ mod n. By Lemma 4, since H_K is uniformly distributed, $(v(M) - v(M'))H_K$ is uniformly distributed in $d\mathbb{Z}_n^l$. Hence, the probability that $0 = (v(M) - v(M'))H_K \mod n$ is equal to one divided by the cardinality of $d\mathbb{Z}_n^l$, which is equal to $(d/n)^l$.

Combining Lemmas 2 and 5 proves Theorem 5. To prove Theorem 1 we need the following extra lemma.

Lemma 6. *Suppose that $v(M) = v(M')$ modulo n, $\sum_{b \in B} M_b = \sum_{b \in B} M'_b$ modulo L, the cardinalities of M and M' are $< L$, and that the multiplicities of M are $< n$. Then $M = M'$.*

Proof. If the cardinalities of M and M' are equal modulo L and $< L$ then

$$\sum_{b \in B} M_b = \sum_{b \in B} M'_b. \tag{5}$$

If all entries of $v(M)$ are $< n$ and $v(M) = v(M')$ modulo n, then

$$M'_b = M_b + \beta_b n, \quad b \in B, \tag{6}$$

for integers $\beta_b \geq 0$. Combining (5) and (6) proves $\sum_{b \in B} \beta_b = 0$, hence, all $\beta_b = 0$. We conclude that $M = M'$.

Now we are ready to prove Theorem 1.

Proof. Let $\mathcal{A}(\mathcal{H}_K)$ be a probabilistic polynomial time (in $m_K \approx 3m$) algorithm with oracle access to $(\mathcal{H}_K, +_{\mathcal{H}_K}, \equiv_{\mathcal{H}_K})$. Then $\mathcal{A}(\mathcal{H}_K)$ can gain knowledge about at most a polynomial number $u(m)$ tuples $[M^i \ ; \ \mathcal{H}_K(M^i)]$ (here $u(.)$ denotes a polynomial). Furthermore, $\mathcal{A}(\mathcal{H}_K)$ can search for a collision among at most a polynomial number $t(m)$ of pairs (M, M'), where M and M' are multisets, $M \neq M'$, and M has multiplicities $< n$. According to Theorem 5, the probability that $\mathcal{A}(\mathcal{H}_K)$ finds a collision is at most

$$t(m)(u(m)^2/2^m + (d/n)^l).$$

Since $\mathcal{A}(\mathcal{H}_K)$ can only compute polynomial sized multisets, the cardinality of the multisets M and M' are $< L \approx 2^m$. This allows us to apply Lemma 6 and conclude that $0 \neq (v(M) - v(M'))$ modulo n. Hence, the greatest common divisor d of n and each of the differences $|M_b - M'_b|$, $b \in B$, is at most $n/2$. This leads to

$$(d/n)^l \leq 2^{-l}.$$

Let $c > 0$ be any number and suppose that $2^{-l} \geq m^{-c}$, or equivalently, $l \leq c \log m$. Notice that each of the differences $|M_b - M'_b|$ is polynomial sized in m, hence, d is polynomial sized in m and there exists a number $e > 0$ such that $d \leq m^e$ for m large enough. This proves

$$(d/n)^l \leq m^{el}/n^l \approx m^{el}/2^m \leq m^{ec \log m}/2^m,$$

which is at most m^{-c} for m large enough. We conclude that the probability that $\mathcal{A}(\mathcal{H}_K)$ finds a collision is at most m^{-c} for m large enough. This proves Theorem 1 for random matrices H_K.

Remark. The theorem also holds for a pseudorandom family of hash functions represented as matrices. Suppose that an adversary can compute a collision with a significant probability of success in the case where a pseudorandom family of hash functions is used. We have just shown that an adversary has a negligible probability of success in the case where random hash functions are used. Hence, with a significant probability of success he is able to distinguish between the use of pseudorandom hash functions and the use of random hash functions. This contradicts the definition of pseudorandomness, see [3] for a detailed proof of a similar result.

B Variants of Additive Hash

A few interesting variants of $\mathcal{H}_K$ exist. Suppose that $v(M) = v(M')$ modulo n and that the multiplicities of M and M' are $< n$. Then clearly $M = M'$. Hence, we do not need Lemma 6 in the proof of Theorem 5. This means that the proof of Theorem 5 does not depend on the cardinalities of M and M' to be equal modulo L. We can remove the cardinality $\sum_{b \in B} M_b$ from the scheme altogether.

For example, for n exponentially large, the cardinalities and in particular the multiplicities of M and M' are $< n$. This proves Corollary 2. An other example is $n = 2$ and both M and M' are sets, which proves the main result of [3].

Secondly, it is possible to replace the random nonce r by a counter that gets incremented on each use of $\mathcal{H}_K$, or by any other value that never repeats itself in polynomial time. This guarantees with probability 1 that the matrix A corresponds to different r^i's (see Lemma 2). This removes the need for a random number generator from the scheme. Moreover, shorter values can be used for r as long as the key is changed when r overflows; this reduces the size of the hash.

If $u = 0$ then the proof of Theorem 5 does not depend on matrix A and its corresponding r^i's. Similarly, if sums of hashes,

$$H_K(0, r) + \sum_{b \in B} M_b H_K(1, b) \mod n,$$

are hidden from the adversary (he knows which multiset M is being hashed, but not the value of the sum of hashes) then we can remove $H_K(0, r)$ from the scheme altogether. As the following corollary shows, complete hiding is not necessary. We can use a pseudorandom permutation to hide sums of hashes.

Corollary 4. *(`Permuted-MSet-XOR-Hash`) The multiset hash corresponding to*

$$\mathcal{H}_{K,K'}(M) = \left[P_{K'}\left(\bigoplus_{b \in B} M_b H_K(1, b) \right) \ ; \ \sum_{b \in B} M_b \mod 2^m \right],$$

where $H_K : \{0, 1\} \times B \to \mathbb{Z}_2^m$ and $P_{K'}$ are randomly selected from a pseudorandom family of hash functions and permutations, is set-collision resistant.

(`Permuted-MSet-Add-Hash`) The multiset hash corresponding to

$$\mathcal{H}_{K,K'}(M) = P_{K'}\left(\sum_{b \in B} M_b H_K(1, b) \mod 2^m \right)$$

where $H_K : \{0, 1\} \times B \to \mathbb{Z}_{2^m}$ and $P_{K'}$ are randomly selected from a pseudorandom family of hash functions and permutations, is multiset-collision resistant.

Notice that the multiset hashes are incremental because $P_{K'}$ is a permutation and, hence, invertible.

Proof. We first consider a random function $P_{K'}$. Suppose that the adversary learns u tuples $[M^i \ ; \ \mathcal{H}_{K,K'}(M^i)]$. As in Lemma 2, the probability that two permuted sums of hashes in the u tuples are equal is at most $u^2/2^m$. If all of them are unequal to one another then matrix AH_K (defined without the part corresponding to the random nonce) is uniformly distributed and not known to the adversary (since $P_{K'}$ is a random function). Hence, the probability that $v(M)H_K = v(M')H_K$ modulo n is statistically independent of the knowledge of the adversary. This can be used instead of Lemma 5 to prove Theorems 5 and 1. This result also holds for a pseudorandom family of permutations $P_{K'}$, see the remark at the end of the proof of Theorem 1 in Appendix A.

C Proof of Collision Resistance of Multiplicative Hash

In the following lemma $\mathcal{A}(.)$ is a probabilistic polynomial time (in $\log q$) algorithm which outputs weights[6] $w_1, \ldots, w_u \in \mathbb{Z}_{q-1}$ for a polynomial number of random inputs $x_1, \ldots, x_u \in GF(q)$ such that $1 = \prod_i x_i^{w_i}$ with probability at least ρ. We show that if such an algorithm exists then we can break the DL problem in $GF(q)$ in polynomial time with probability at least ρ.

Lemma 7. *Let $\mathcal{A}(.)$ be a ppt algorithm such that there exists a number c such that for $u \leq (\log q)^c$,*

$$Prob \left\{ \begin{array}{l} (x_i \leftarrow GF(q))_{i=1}^u, (w_i \in \mathbb{Z}_{q-1})_{i=1}^u \leftarrow \mathcal{A}(x_1, \ldots, x_u) : \\ 1 = \prod_i x_i^{w_i}, \quad \exists_i w_i \neq 0, \quad \forall_i |w_i| \leq (\log q)^c \end{array} \right\} \geq \rho. \tag{7}$$

Let g be a generator of $GF(q)$. Then there exists a probabilistic polynomial time (in $\log q$) algorithm $\mathcal{A}'(.)$ such that

$$Prob\{y \leftarrow GF(q), x \leftarrow \mathcal{A}'(y) : y = g^x\} \geq \rho/(\log q)^c.$$

In words, given a random $y \in GF(q)$, we are able to find the discrete log of y in $GF(q)$ with probability at least $\rho/(\log q)^c$.

Proof. Let $y \leftarrow GF(q)$. Select a polynomial number u of random elements $r_1, \ldots, r_u$ in $\mathbb{Z}_{q-1}$ and $j \in \{1, \ldots, u\}$ and compute

$$x_j = y g^{r_j} \text{ and } x_i = g^{r_i} \text{ for } i \neq j.$$

Compute $(w_1, \ldots, w_u) \leftarrow \mathcal{A}(x_1, \ldots, x_u)$. Since by construction the x_is have been chosen uniformly at random, we know that with probability at least ρ the weights $w_1, \ldots, w_u \in \mathbb{Z}_{q-1}$ are computed such that they are not all equal to zero, $|w_j| \leq (\log q)^c$, and

$$1 = \prod_i x_i^{w_i} = y^{w_j} g^{\sum_i r_i w_i}. \tag{8}$$

Since the u inputs are in random order, the probability that $w_j \neq 0$ is at least

$$1/u \geq (\log q)^{-c}.$$

Suppose that $w_j \neq 0$. Let d be the greatest common divisor between w_j and $q-1$. Then[7] w_j/d is invertible in $\mathbb{Z}_{q-1}$. By using the Chinese remainder theorem (assuming that we know the factorization of $q-1$), we are able to compute the inverse of w_j/d in $\mathbb{Z}_{q-1}$ in polynomial time. Denote this inverse by w_j'. From (8) we infer that

$$y^d = g^{-w_j' \sum_i r_i w_i}.$$

[6] Not all equal to zero and each of them bounded by a polynomial number.

[7] Division / denotes division over integers, not over $\mathbb{Z}_{q-1}$ (since d has no inverse in $\mathbb{Z}_{q-1}$, we can not divide w_j by d in $\mathbb{Z}_{q-1}$).

Notice that if $y^d = g^s$ and $y = g^t$, then $g^{dt} = g^s$, that is $dt = s$ modulo $q-1$. Recall that d divides $q-1$. For this reason d must also divide s. Let $d' = (q-1)/d$ and $s' = s/d$. Both can be computed in polynomial time as we have shown. Now y can be expressed as one of the roots

$$y = g^{s'+jd'},$$

where $0 \le j \le d-1$. Since $d \le |w_j| \le (\log q)^c$, each of the roots can be checked in polynomial time. This proves the lemma.

The DL assumption states that for all ppt algorithms $\mathcal{A}(.)$, any number c, and Q large enough,

$$Prob \left\{ \begin{array}{l} q \ge Q \text{ is a prime power, } g \text{ generates } GF(q), \\ y \leftarrow GF(q), x \leftarrow \mathcal{A}(q,y) \end{array} : y = g^x \right\} \le (\log q)^{-c}.$$

We are ready to prove Theorem 2.

Proof. Suppose that there exists a number c and a probabilistic polynomial time algorithm $\mathcal{B}(H)$, which runs in time $u = (\log q)^c$, with access to a random oracle H which outputs with probability $\rho \ge 1/u$ a collision M for M'. That is, $M \ne M'$, M and M' are polynomial sized $< u$, and

$$\mathcal{H}(M) = \prod_{b \in B} H(b)^{M_b} = \prod_{b \in B} H(b)^{M'_b} = \mathcal{H}(M').$$

This means that

$$1 = \prod_{b \in B} H(b)^{M_b - M'_b},$$

there is a polynomial number M_b's and M'_b's unequal to zero, for all $b \in B$ the absolute value $|M_b - M'_b| < u$ is polynomial sized, and there exists a $b \in B$ such that $M_b - M'_b \ne 0$.

Let $\mathcal{C}$ be an algorithm that goes from $GF(q)^u$ to $B \to GF(q)$, where $B \to GF(q)$ denotes the set of oracles with inputs in B and outputs in $GF(q)$. $\mathcal{C}$ is chosen such that $\mathcal{C}(x_1, \dots, x_u)$ returns x_1 when it is called for the first time on some input y_1, x_2 when it is called for the first time on some input y_2 different from y_1, and so on.

When $x_1, \dots, x_u$ are chosen randomly, $\mathcal{C}(x_1, \dots, x_u)$ cannot be distinguished from a random oracle by $\mathcal{B}$ because $\mathcal{B}$ cannot query $\mathcal{C}$ more than u times. Therefore, if we let $\mathcal{A}$ be the composition of $\mathcal{B}$ and $\mathcal{C}$, $\mathcal{A}$ is able to find a collision for $\mathcal{H}$ with probability ρ when its inputs are chosen uniformly at random. Moreover, $\mathcal{A}$ is a ppt algorithm satisfying (7), so by Lemma 7, $\mathcal{A}$ can break the discrete log problem in $GF(q)$ in polynomial time with probability at least $\rho/(\log q)^c \ge (\log q)^{-2c}$. This contradicts the DL assumption. So $\mathcal{B}$ does not exist, which proves multiset-collision resistance.

Because oracle access to H is stronger than oracle access to $\mathcal{H}$, this proves Theorem 2 when H is a random oracle. The result carries over to poly-random functions because they are indistinguishable from random functions by ppt algorithms.

Remark. Supposing that H is a random oracle is a strong assumption. Compared to the `MSet-XOR-Hash` and `MSet-Add-Hash` we do not need a secret key (as the seed of a pseudorandom family of hash functions) at all. We refer to [5] for a discussion into what extent the random oracle assumption can be met in practice.

D Proof of Collision Resistance of Vector Additive Hash

If r is a fixed constant in the `MSet-Add-Hash`, then we are again vulnerable for the attack described for the `MSet-XOR-Hash`, where r is a fixed constant. The main difference is that the attack is not modulo $n = 2$ but modulo $n = 2^m$. This means that the linear combination may lead to a collision with large multiplicities. This would give a non-polynomial sized collision and does not defeat the multiset collision resistance. It turns out that this problem is related to a weighted knapsack problem (see also [4]). In this sense `MSet-Add-Hash` remains multiset collision resistant, even if the pseudorandom family of hash functions H_K is replaced by a single random function avoiding the use of a secret key as in `MSet-Mu-Hash`.

The weighted knapsack (WK) assumption is defined as follows. For all ppt algorithms $\mathcal{A}(.)$, any number c, q large enough, and $u \leq (\log q)^c$,

$$Prob \left\{ \begin{array}{l} (x_i \leftarrow \mathbb{Z}_q)_{i=1}^u, (w_i \in \mathbb{Z}_q)_{i=1}^u \leftarrow \mathcal{A}(x_1, \ldots, x_u) : \\ 0 = \sum_i w_i x_i \mod q, \quad \exists_i w_i \neq 0, \quad \forall_i |w_i| \leq (\log q)^c \end{array} \right\} \leq (\log q)^{-c}.$$

Notice the resemblance with (7), where multiplication in $GF(q)$ is now replaced by addition modulo q (where q can be any integer and does not need to be a prime power). It remains unclear to what extent Ajtai's work [1] relates this problem to the *worst-case* shortest vector problem. It is an open problem whether to believe in the WK assumption.

Let $H : B \rightarrow \mathbb{Z}_q$ be a poly-random function. We define

$$\mathcal{H}(M) = \sum_{b \in B} M_b H(b) \mod q, \tag{9}$$

$\equiv_{\mathcal{H}}$ to be equal to $=$, and $+_{\mathcal{H}}$ to be addition modulo q (q plays the role of 2^m in `MSet-Add-Hash`). The proof of the next theorem is similar to the proof of Theorem 2 in Appendix C.

Theorem 6. *Under the WK assumption, $(\mathcal{H}, +_{\mathcal{H}}, \equiv_{\mathcal{H}})$ as defined in (9) is multiset collision resistant.*

For completeness, we introduce a multiset hash corresponding to parameters $n = 2^{\sqrt{m}}$ and $l = \sqrt{m}$ (see Section 4). Let $H : B \rightarrow \mathbb{Z}_n^l$ be a poly-random function. Now, we define

$$\mathcal{H}(M) = \sum_{b \in B} M_b H(b) \mod n,$$

$\equiv_{\mathcal{H}}$ to be equal to $=$, and $+_{\mathcal{H}}$ to be vector addition modulo n. Theorem 6 holds again if we modify the WK assumption by replacing $x_i \leftarrow \mathbb{Z}_q$ by $x_i \leftarrow \mathbb{Z}_n^l$, $w_i \in \mathbb{Z}_q$ by $w_i \in \mathbb{Z}_n$, and q by n. The main difference is that the x_i's are vectors of length $l = \sqrt{m}$. According to [8, Sections 2.1 and 2.2][8], if there is a ppt solving the modified WK problem (that is it contradicts the modified WK assumption) then, by Ajtai's theorem [1], there is a probabilistic polynomial (in l) algorithm which, for *any* lattice $\mathcal{L}$ in $\mathbb{R}^l$, given an arbitrary basis of $\mathcal{L}$, approximates (up to a polynomial factor in l) the length of the shortest vector in $\mathcal{L}$. This proves Theorem 3. The worst-case shortest vector problem is believed to be hard, see [8] for more discussion.

E Proof of Improved Offline Checker

In this appendix, we prove Theorem 4.

Proof. Suppose the RAM does not behave like valid RAM (i.e. the data value that the checker reads from an address is not the same data value that the checker had most recently written to that address). We will prove that $W \neq R$.

Consider the `put` and `get` operations that occur on an address as occurring on a timeline. To avoid confusion with the values of Timer, we express this timeline in terms of processor cycles. Let x_1 be the cycle of the first incorrect `get` operation. Suppose the checker reads the pair (v_1, t_1) from address a at x_1. If there does not exist a cycle at which the checker writes the pair (v_1, t_1) to address a, then $W \neq R$ and we are done.

Suppose there is a cycle x_2 when the checker first writes (v_1, t_1) to address a. Because of line 3 in the `get` operation, the values of time stamps of all of the writes to a after x_1 are strictly greater than t_1. Because the time stamps at x_1 and x_2 are the same, and since `put` operations and `get` operations do not occur on the same cycle, x_2 occurs before x_1 ($x_2 < x_1$). Let x_3 be the cycle of the first read from a after x_2. Notice that x_1 is a read after x_2, so $x_1 \geq x_3$. If x_1 were equal to x_3, then the data value most recently written to a, i.e. v_1, would be read at x_1. This contradicts the assumption that x_1 is an incorrect read. Therefore, $x_1 > x_3$.

Because the read at cycle x_1 is the first incorrect read, the read at cycle x_3 is a correct read. So the read at x_3 reads the same pair that was written at x_2. Again, because of line 3 in the `get` operation, the values of time stamps of all the writes to a after x_3 are strictly greater than t_1. Therefore, (v_1, t_1) cannot be written after x_3. Because x_2 is the first cycle on which (v_1, t_1) is written to a, (v_1, t_1) cannot be written before x_2. Because x_3 is the first read from a after x_2, and two writes to an address always have a read from that address between them, (v_1, t_1) cannot be written between x_2 and x_3. Therefore, the pair (v_1, t_1) is written only once, but it is read at x_1 and x_3. Therefore, $W \neq R$.

[8] Notice that the matrix with columns x_i is in $\mathbb{Z}_n^{l \times u}$ and that the vector with entries w_i is unequal to zero and has Euclidean norm polynomial in $l = \sqrt{m}$.