# On the Impossibility of Merkle Merge Homomorphism

Yuzhe Tang[†]
Syracuse University
Syracuse, NY 13244
[†]ytang100@syr.edu

## 1. INTRODUCTION

This work considers a theoretic problem of merging the digests of two ordered lists "homomorphically." This problem has potential applications to efficient and verifiable data outsourcing, which is especially desirable in the public cloud computing where the cloud is not trustworthy. We consider the case of `merge`-sort as it is fundamental to many cloud-side operations, such as database join [4], data maintenance [1], among others.

Informally, a `merge`-homomorphic digest enables that the digest of an ordered list, merged from two sublists, is computable from the digests of the two sublists. We present the formal definition of a `merge`-homomorphic digest (§ 2).

We then examine the feasibility of using Merkle hash tree or MHT [3] to construct the `merge`-homomorphic digest (§ 3). Our theoretic result is that we proved the impossibility of `merge`-homomorphism for MHT (§ 3.2) by contradiction to the definition of collision-resistant hashes.

This negative result is useful to understanding the limitations for designing a `merge`-homomorphic digest and might shed lights for a correct construction in the future.

## 2. MERGE-HOMOMORPHIC DIGEST

DEFINITION 2.1 (ORDER-SENSITIVE DIGEST). *Let $\mathbb{M}_O$ be the domain of records; the length of record is bound by $k(\mathbb{M}_O)$. Let $\mathbb{M}_S$ be the domain of lists of records from $\mathbb{M}_O$ and with order sensitivity – to the same set of records, lists of different orderings differ. A digest function $D(L)$ on message space $\mathbb{M}_S$ is a collision-resistant order-sensitive digest function (or CROD), if the collision resistance property applies to the order-sensitive lists in $\mathbb{M}_S$; that is, for any list $L \in \mathbb{M}_S$, it is computationally unfeasible to find a different list $L'$ such that $D(L) = D(L')$.*

For instance, a hash chain constructed on top of an ordered list is a CROD. And so is a Merkle hash binary tree constructed on an ordered list in $\mathbb{M}_S$.

DEFINITION 2.2 (MERGE-HOMOMORPHIC DIGEST). *Consider a CROD $D(L)$ on domain $\mathbb{M}_S$. Let $\oplus$ be a binary function on elements in the domain of $D(\cdot)$. CROD $D(L)$ has the `merge`-homomorphism property (or is `merge`-homomorphic) if $\forall L_1, L_2 \in \mathbb{M}_S$, the following equation always holds:*

$$D(\texttt{merge}(L_1, L_2)) = D(L_1) \oplus D(L_2) \quad (1)$$

*Here, $\texttt{merge}(L_1, L_2)$ is a merge operation of two ordered lists $L_1$ and $L_2$.*

## 3. THE CASE OF MHT FOR MERGE-HOMOMORPHISM

We first present the preliminary on Merkle hashes tree before formally proving our impossibility results.

### 3.1 Preliminary: Merkle hash tree

DEFINITION 3.1 (COLLISION-RESISTANT HASH). *A cryptographic hash functions is collision resistant, which means that for* any *$u$, it is computationally unfeasible to find a different value $u'$ such that $H(u) = H(u')$. In here, we consider a cryptographic hash $H(\cdot)$ takes a variable-length input $u$ and generates a fixed-length output $z = H(u)$.*

DEFINITION 3.2. *A Merkle hash tree (MHT) is constructed by the following rules: for any two lists, $L < L'$,[1] $MH(L \| L') = H(MH(L) \| MH(L'))$. Here $MH(\cdot)$ denotes the root hash of the MHT, $H(\cdot)$ is a cryptographic hash function, and $\|$ is concatenation on sets or Merkle hash strings.*

### 3.2 Impossibility of Merkle `merge`-homomorphism

LEMMA 3.3. *Cryptographic hash can not be collision free to all input values. That is, there must exist value $u$, such that there exist value $u'$ and $H(u) = H(u')$.*

The proof sketch of Lemma 3.3 is due to Definition 3.1; the input space of a hash is much larger than the output space of the hash; hence there must be collisions.

THEOREM 3.4. *An MHT can not be `merge`-homomorphic. That is, there exists no polynomial-time operation $\oplus$ such that $MH(\texttt{merge}(L_1, L_2)) = MH(L_1) \oplus MH(L_2)$.*

PROOF. We prove the theorem by contradicting the `merge`-homomorphism to Lemma 3.3.

[1]This means the largest element in $L$ is smaller than the smallest element in $L'$.

Due to Lemma 3.3, there exist a value $x$ with collision, that is, there is another value $y$ such that $H(x) = H(y)$. Without loss of generality, we assume $x < y$. We denote by $L_1$ and $L_2$ two single-element lists respectively containing $x$ and $y$; $L_1 = \{x\}$ and $L_2 = \{y\}$. Therefore:

$$MH(L_2) = H(y) = H(x) = MH(L_1)$$

Assuming there is an MHT with `merge`-homomorphism:

$$\begin{aligned} MH(\texttt{merge}(L_1, L_2)) &= MH(L_1) \oplus MH(L_2) \\ &= MH(L_1) \oplus MH(L_1) \\ &= MH(\texttt{merge}(L_1, L_1)) \\ &= MH(L_1) = H(x) \end{aligned} \tag{2}$$

On the other hand, we have:

$$\begin{aligned} MH(\texttt{merge}(L_1, L_2)) &= MH(\{x, y\}) \\ &= H(H(x) \| H(y)) \\ &= H(H(x) \| H(x)) \end{aligned} \tag{3}$$

By combining Equation 2 and Equation 3, we have $H(x) = H(H(x)|H(x))$, hence another collision.[2]

Thus, from the existence of collision between $x$ and $y$, we find another collision of $x$ whose value can be easily derived from $x$, $H(x)|H(x)$. Because there must be at least one value with collision (Lemma 3.3), we find a contradiction to hash collision-resistance (Definition 3.1). □

We note our result is generic and applies to other "integrity tree" [2] which, comparing Merkle hash tree, use other digest functions (e.g. MAC) than hash.

## Acknowledgement

The author would thank Dr. Hugo Krawczyk for discussion in the early stage of the work.

## 4. REFERENCES

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data (awarded best paper!). In *OSDI*, pages 205–218, 2006.

[2] R. Elbaz, D. Champagne, C. H. Gebotys, R. B. Lee, N. R. Potlapally, and L. Torres. Hardware mechanisms for memory authentication: A survey of existing techniques and engines. *Trans. Computational Science*, 4:1–22, 2009.

[3] R. C. Merkle. A certified digital signature. In *Proceedings on Advances in Cryptology*, CRYPTO '89, 1989.

[4] Y. Zhang, J. Katz, and C. Papamanthou. Integridb: Verifiable SQL for outsourced databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1480–1491, 2015.

[2] It is easy to see $x \neq H(x)|H(x)$ as they are on different domains: By the definition of cryptographic hash, the input domain ($x$) can be arbitrarily larger than the output domain ($H(x)$).