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 theoretic 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 [6], data maintenance [3], 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 [5] to construct the `merge`-homomorphic digest (§ 3). Our theoretic result is that we proved the impossibility of `merge`-homomorphism for MHT (§ 3.1) 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 SIGNATURE

A `merge`-homomorphic signature scheme for message space $\mathbb{M}$ consists of four polynomial-time algorithms as below. Our formulation follows those in homomorphic MAC [1] and p-homomorphic signature [2].

- $\mathsf{KeyGen}(1^k) \rightarrow sk, pk$: the key generation algorithm takes as input 1^k (k is security parameter), and outputs a key pair including public key pk and secret key sk.
- $\mathsf{Sign}(sk, L_1) \rightarrow \delta_1, \mathsf{state}$ and $\mathsf{Sign}(sk, L_2, \mathsf{state}) \rightarrow \delta_2$: We only consider the simplest case of signing up to two ordered lists; It can be naturally extended to the case of more than two ordered lists. To L_1, the algorithm takes as input the secret key sk, an ordered lists L_1 from message space $\mathbb{M}$. The algorithm produces as output the signature δ_1 on L_1, and a state. To L_2, the algorithm differs in taking the state as input and producing only $delta_2$ as output.
- $\mathsf{SignMerge}(pk, L_1, L_2, \delta_1, \delta_2, L) \rightarrow \{\delta_{12}, \perp\}$: the algorithm takes as input the public key pk, two ordered lists L_1 and L_2 from message space $\mathbb{M}$ with their corresponding signatures δ_1 and δ_2, and a list L. If $L = \mathtt{merge}(L_1, L_2)$, the algorithm outputs the signature δ_{12} on L_{12}; otherwise, it outputs $\perp$.
- $\mathsf{Verify}(pk, L, \delta) \rightarrow \{0, 1\}$: the algorithm takes as input public key pk, an ordered list L from message space $\mathbb{M}$ with its signature δ, and outputs a binary indicating whether the signature is valid (1 for valid signature and 0 otherwise).

Correctness. We require that all key pairs (pk, sk) generated by $\mathsf{KeyGen}(1^k)$ and for all $L_1, L_2, L \in \mathbb{M}$ we have:

- if $\mathtt{merge}(L_1, L_2) = L$, then $\mathsf{SignMerge}(pk, L_1, L_2, \delta_1, \delta_2) \neq \perp$ and
- for all δ_1 such that $\mathsf{SignMerge}(pk, L_1, L_2, \delta_1, \delta_2, L) \rightarrow \delta_{12}$, we have $\mathsf{Verify}(pk, L, \delta_{12}) = 1$.

Security: Unforgeability. We define the security for `merge`-homomorphic signature. We use the basic notion of existential unforgeability with regards to adaptive chosen message attacks (CMA) [4]. Roughly, the idea is to allow the attacker to obtain the signature on arbitrary ordered lists of her choice (analogous to a chosen message attack on signatures). The unforgeability states the attacker should be unable to produce a valid tuple (L, δ_{12}) where $L \neq \mathtt{merge}(L_1, L_2)$. Formally,

Attack game. Let $\mathcal{T} = (\mathsf{Sign}, \mathsf{SignMerge}, \mathsf{Verify})$ be a `merge`-homomorphic signature scheme. We define the security of $\mathcal{T}$ using the following game between a challenger and a PPT adversary $\mathcal{A}$.

Setup. The challenger runs KeyGen and obtains key-pair sk, pk.

Queries. $\mathcal{A}$ adaptively submits a series of queries where the i-th query is (L_1^i, L_2^i); L_1^i and L_2^i are two ordered lists chosen from message space $\mathbb{M}$. For each query, the combination of (L_1^i, L_2^i) is unique. To respond to query (L_1^i, L_2^i), the challenger sends back to $\mathcal{A}$: $\delta_1^i = \mathsf{Sign}(L_1^i), \delta_2^i = \mathsf{Sign}(L_2^i)$.

Output. The adversary $\mathcal{A}$ outputs (L, δ).

The adversary wins the security game if $\mathsf{Verify}(pk, L, \delta) = 1$, and $\forall i, L \neq \mathtt{merge}(L_1^i, L_2^i)$.

The advantage NC-adv$[\mathcal{A}, \mathcal{T}]$ of adversary $\mathcal{A}$ w.r.t. $\mathcal{T}$ is defined to be the probability that $\mathcal{A}$ wins the security game.

DEFINITION 2.1. *A `merge`-homomorphic signature scheme $\mathcal{T}$ is secure if for all polynomial time adversary $\mathcal{A}$ the quantity NC-adv$[\mathcal{A}, \mathcal{T}]$ is negligible.*

Conference Location

3. CANDIDATE CONSTRUCTION BY MERKLE HASH TREE

We first describe a conjectured property of a Merkle hash tree (MHT), and then describe the possible construction of merge-homomorphic signature using MHT.

DEFINITION 3.1. *Let $\oplus$ be a binary function on elements in the domain of Merkle hash digests. Let O be the domain of records with ordering; the length of record is bound by $k(O)$. Let $\mathbb{M}_S$ be the domain of possible ordered lists of records from O. We say an MHT on domain $\mathbb{M}_S$ has the merge-homomorphism property (or is merge-homomorphic) if $\forall L_1, L_2 \in O$, the following equation always holds:*

$$MH(merge(L_1, L_2)) = MH(L_1) \oplus MH(L_2) \qquad (1)$$

Here, $MH(L_1)$ denotes a Merkle hash digesting ordered list L_1. $\texttt{merge}(L_1, L_2)$ is a merge operation of two ordered lists L_1 and L_2.

We now consider the construction of the digest between a verifier V and a prover P. The construction is below:

- V.KeyGen$(1^k) \rightarrow sk, pk$ is by the standard public-private key generation.
- V.Sign$(sk, L_1) \rightarrow \delta_1$, state:
  1. $rh_1 = MH(L_1)$
  2. state.add(rh_1)
  3. $\delta_1 = Sg_{sk}(rh_1)$, where $Sg_{sk}(\cdot)$ is the signing function in a digital signature.
  4. output δ_1, state
- V.Sign$(sk, L_2,$ V.state$) \rightarrow \delta_2$:
  1. $rh_2 = MH(L_2)$
  2. $rh_1 \rightarrow$ V.state
  3. $rh_{12} = rh_1 \oplus rh_2$
  4. V.state.add(rh_{12}, rh_2)
  5. output $\delta_2 = \{Sg_{sk}(rh_2), Sg_{sk}(rh_{12})\}$
- P.SignMerge$(pk, L_1, L_2, \delta_1, \delta_2, L) \rightarrow \{\delta_{12}, \perp\}$:
  1. if$(L = \texttt{merge}(L_1, L_2))$ output $\delta_{12} = Sg_{sk}(rh_{12})$
  2. else outputs $\perp$
- P.Verify$(pk, L, \delta) \rightarrow \{0, 1\}$:
  1. $Vf(\delta_{12}, MH(L_{12}))$ where Vf is the verification function in a digital signature.

Correctness. can be easily established.

Unforgeability. requires that if P.Verify$(pk, L, \delta) = 1$ then $L = \texttt{merge}(L_1, L_2)$. The unforgeability can be proved by: 1) the unforgeability of a digital signature which ensures that when $Vf(\delta_{12}, MH(L_{12})) = 1$, $MH(L_{12}) = rh_{12} = rh_1 \oplus rh_2$ and 2) the following equation:

$$\begin{aligned} MH(L_{12}) &= rh_{12} = rh_1 \oplus rh_2 \\ &= MH(L_1) \oplus MH(L_2) \\ &= MH(\texttt{merge}(L_1, L_2)) \end{aligned}$$

Due to the oneway-ness of $MHT(\cdot)$, $L_{12} = \texttt{merge}(L_1, L_2)$.

It can be seen the correctness of using MHT for merge-homomorphic signature depends on the existence of a merge-homomorphic MHT. Unfortunately, a merge-homomorphic MHT does not exist. We formally present our theoretical result.

3.1 Impossibility of Merkle merge-homomorphism

THEOREM 3.2. *The is no Merkle tree that can instantiates the Merkle merge-homomorphism defined in Definition 3.1. In other words, there is no operation $\oplus$ such that $MH(merge(L_1, L_2)) = MH(L_1) \oplus MH(L_2)$.*

PROOF. We prove the theorem by contradicting the merge-homomorphism to the following facts:

G1. The construction of an MHT, that is, for any two lists, $L < L'$,[1] $MH(L \| L') = H(MH(L) \| MH(L'))$. Here, $H(\cdot)$ is a cryptographic hash function, and $\|$ is concatenation on sets or Merkle hash strings..

G2. By definition, 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 $z = H(u)$.

G3. 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 $H(u) = H(u')$ – By definition, the input space of a hash is much larger than the output space of the hash; hence there must be collision.

We consider a value x with collision, that is, there is 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:

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

On the other hand, we have:

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

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

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 is easy to derive from x, $H(x)|H(x)$. Because there must be at least one value with collision (G3), we find a contradiction to collision resistance (G2). $\square$

4. REFERENCES

[1] S. Agrawal and D. Boneh. Homomorphic macs: Mac-based integrity for network coding. In *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings*, pages 292–305, 2009.

[2] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, A. Shelat, and B. Waters. Computing on authenticated data. *J. Cryptology*, 28(2):351–395, 2015.

[3] 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.

[4] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

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

[6] 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 ($H(x)$).