Delta-XMSS: Incremental State Optimization for Post-Quantum Hash-Based Signatures

Antônio Augusto Tavares Ribeiro André[1], Routo Terada[2], Victor Takashi Hayashi[3], Bryan Kano Ferreira[1]

[1] Instituto de Tecnologia e Liderança (Inteli)
São Paulo – SP – Brazil

[2]Instituto de Matemática e Estatística (IME) – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

[3]Escola Politécnica (EPUSP) – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

antonio.andre@sou.inteli.edu.br, rt@ime.usp.br

victor.hayashi@usp.br, bryan.ferreira@prof.inteli.edu.br

Abstract. *EXtended Merkle Signature Scheme (XMSS) is a hash-based digital signature scheme whose security relies on its underlying hash function and is therefore unaffected by Shor's algorithm. However, one of the challenges faced by XMSS is the size of its signatures: an authentication path is transmitted along each signature to reconstruct the Merkle tree and a Winternitz One-Time Signature Plus (WOTS+) signature. To solve this problem, approaches like Buchmann, Dahmen, Schneider (BDS) optimize the generation of the authentication path along the signatures of the WOTS+ leaf but they still transmit the entire path. For this, we propose a technique to reduce the authentication path by delta-encoding two consecutive paths and transmitting only the nodes that change between them. We show that our solution reduces the authentication path size from $h' \cdot n$ to $(\nu(idx) + 1) \cdot n$ bytes per signature, achieving reductions of up to 90% in the path, corresponding to up to 11.5% of the complete signature (21.6% for $h' = 20$). Encoding and decoding overhead remain under 80 CPU cycles for the most frequent signing indices ($\nu \leq 5$), negligible relative to the XMSS signing operation itself (median of 5,711,700 cycles).*

Keywords: XMSS, Delta encoding, Authentication path, Post-quantum cryptography.

1. Introduction

The evolution of quantum computing has motivated the search for cryptographic schemes capable of resisting quantum attacks such as Shor's algorithm [Shor 1994], which breaks Rivest–Shamir–Adleman (RSA) [Rivest et al. 1978] in polynomial time. Hash-based signature schemes such as the eXtended Merkle Signature Scheme (XMSS) are natural candidates, as their security relies solely on the underlying hash function and is therefore unaffected by Shor's algorithm [Buchmann et al. 2011]. XMSS, introduced by Buchmann, Dahmen, and Hülsing (2011), and recommended in NIST SP 800-208

[National Institute of Standards and Technology 2020] as a stateful hash-based signature is based on a set of Merkle trees [Merkle 1990] to structure digital signatures using Winternitz One-Time Signature Plus (WOTS+) leaves and authentication paths that connect these leaves to the root of the tree [Buchmann et al. 2011].

Despite its robustness, one of the challenges faced by XMSS is the size of its signatures. Along with each signature, an authentication path is transmitted to reconstruct the WOTS+ leaf up to the root. When signing consecutive messages within the same tree, the indices of the leaves increase sequentially, causing different authentication paths to exhibit high similarity. Even so, implementations such as the one presented in RFC 8391 [Hülsing et al. 2018] still transmit the entire path with each new signature, repeating data and wasting storage. This redundancy in data transmission creates an opportunity to implement optimization techniques, such as delta encoding [Hunt and McIlroy 1976], which allows transmitting only the parts that change between consecutive signatures within the same tree.

2. Related Work

Hash-based signatures have been extensively studied in post-quantum cryptography. Buchmann, Dahmen, and Schneider introduced the BDS algorithm [Buchmann et al. 2008] which optimizes the generation of the authentication path. While BDS reduces computational overhead, it does not reduce the cost of transmission, which still transmits the whole authentication path with each signature.

Delta encoding has been applied to reduce transmission overhead in constrained environments. Jindal et al. [Jindal et al. 2022] proposed a model for IoT data stream transmission using delta encoding for compression, achieving data reduction of up to 37.59% and reducing average transmission time and power consumption. Delta encoding has also been applied in other constrained settings, such as event-based ECG classification [Jobst et al. 2020]. Modern delta encoding algorithms are based on a foundation algorithm for differential file comparison [Hunt and McIlroy 1976].

To the best of our knowledge, based on a survey of the literature (e.g., Google Scholar and IEEE Xplore), no other work has applied delta encoding to reduce the transmission of authentication paths on an XMSS scheme. Delta-XMSS explores the gap in transmission overhead as a consequence of the structural overlap of consecutive paths.

3. Preliminaries

Notation and Hash Function. Let n be the size of the hash function output, h' be the height of the Merkle tree. For $0 \leq j \leq h'$ and $0 \leq k < 2^{h'-j}$, let $n_{j,k}$ denote the node at level j and position k in the tree. Let $0 \leq idx < 2^{h'}$ denote the index of a leaf. At level j, the index of the node corresponding to this leaf is:

$$k_j = \left\lfloor \frac{idx}{2^j} \right\rfloor$$

The sibling (neighbor) node at level i is therefore $n_{i,\, k_i \oplus 1}$, where $\oplus 1$ flips the least significant bit.

The internal nodes of the tree are computed via a collision-resistant hash function $H : \{0,1\}^{2n} \to \{0,1\}^n$, where each node is recursively defined as:

$$n_{j,k} = H(n_{j-1,\,2k} \parallel n_{j-1,\,2k+1})$$

H at XMSS additionally takes a public seed PK.seed and an address parameter $ADRS$ for domain separation but does not affect our delta construction [Hülsing et al. 2018].

Authentication Path. The authentication path $\mathcal{A}_{idx}$ of a leaf is the sequence of sibling nodes to compute the tree from the leaf to the root:

$$\mathcal{A}_{idx} = \left(n_{0,\,k_0\oplus 1},\ n_{1,\,k_1\oplus 1},\ \ldots,\ n_{h'-1,\,k_{h'-1}\oplus 1}\right)$$

XMSS Signature. Let SIG_{idx} be the XMSS signature composed by the idx of the node assigned, a randomization value r, a WOTS+ signature and a $\mathcal{A}_{idx}$. Implementations such as the one defined in RFC 8391 append the signed message m to form $SM_{idx} = SIG_{idx} \| m$ [Hülsing and Rijneveld 2018].

Consecutive Paths. Let $\nu(idx)$ be the number of trailing ones in the binary representation of idx. The authentication paths $\mathcal{A}_{idx}$ and $\mathcal{A}_{idx+1}$ share the same nodes from $\nu(idx)+1$ to $h'-1$ and differ at levels 0 through $\nu(idx)$.

Delta. The delta between two consecutive paths are the nodes from $\mathcal{A}_{idx+1}$ that differ from $\mathcal{A}_{idx}$ and we define as:

$$\Delta_{idx} = \left(n_{0,\,k_0\oplus 1},\ \ldots,\ n_{\nu(idx),\,k_{\nu(idx)}\oplus 1}\right)$$

where it takes the values of the nodes that differ from the levels 0 through $\nu(idx)$.

4. Delta-XMSS Construction

4.1. Overview

The general idea of the delta implementation on the authentication paths is to store $\mathcal{A}_{idx}$ in cache and use it to compute Δ_{idx}, the set of nodes that differ between $\mathcal{A}_{idx}$ and $\mathcal{A}_{idx+1}$. The receiver that stores $\mathcal{A}_{idx}$ in cache will apply Δ_{idx} to reconstruct $\mathcal{A}_{idx+1}$ and replace it at the cached $\mathcal{A}_{idx}$. Only $(\nu(idx)+1)\cdot n$ instead of $h'\cdot n$ bytes are transmitted.

4.2. Encoding

The encoding algorithm takes SM_{idx+1} as input, extracts $idx+1$ and subtracts 1 to get the idx used to compute $\nu(idx)$ and returns Δ_{idx}, the nodes that differ from $\mathcal{A}_{idx}$ and $\mathcal{A}_{idx+1}$.

Algorithm 1: DeltaEncode

Input: SM_{idx+1}
Output: Δ_{idx}
Extract $idx+1$ from SM_{idx+1};
Let $idx \leftarrow (idx+1)-1$;
Compute $v \leftarrow \nu(idx)$;
Let $auth$ be the authentication path extracted from SM_{idx+1};
for $j \leftarrow 0$ **to** v **do**
  $\Delta_{idx}[j] \leftarrow auth[j]$;
return Δ_{idx};

4.3. Decoding

The decoding algorithm takes the output Δ_{idx} from Algorithm 1 and the stored $\mathcal{A}_{idx}$ as input to create $\mathcal{A}_{idx+1}$. The idx is used to compute $\nu(idx)$.

Algorithm 2: DeltaDecode

Input: $\mathcal{A}_{idx}$, Δ_{idx}, idx
Output: $\mathcal{A}_{idx+1}$
Compute $v \leftarrow \nu(idx)$;
$\mathcal{A}_{idx+1} \leftarrow \mathcal{A}_{idx}$;
for $j \leftarrow 0$ **to** v **do**
  $\mathcal{A}_{idx+1}[j] \leftarrow \Delta_{idx}[j]$;
return $\mathcal{A}_{idx+1}$;

4.4. Delta encoding

Lemma 1. *Let h' be the height of the Merkle Tree, n the node size in bytes, and $0 \leq idx < 2^{h'} - 1$ a leaf index. Let $v = \nu(idx)$ where $\nu(idx)$ is the number of trailing ones in idx. Let $\mathcal{A}_{idx}$ be the authentication path stored in cache, and Δ_{idx} the output of DeltaEncode. Then DeltaDecode$(\mathcal{A}_{idx}, \Delta_{idx}, idx) = \mathcal{A}_{idx+1}$*

Proof. By the definition of consecutive paths (Section 3) , two consecutive paths $\mathcal{A}_{idx}$ and $\mathcal{A}_{idx+1}$ satisfy:

- for every j where $v < j \leq h' - 1$, the sibling node at level j is the same in both paths.
- for every j where $0 \leq j \leq v$, the sibling node at level j differs between both paths.

Following the standard structure of a Merkle Tree at RFC-8391 [Hülsing et al. 2018], the bit j at the binary form of idx determines whether the ancestor of leaf idx at level j is a left child (if $\lfloor idx/2^j \rfloor \equiv 0 \pmod 2$), or is a right child (if $\lfloor idx/2^j \rfloor \equiv 1 \pmod 2$). v determines the number of trailing ones in idx and the total of levels j where the ancestor is the right child. When we calculate $idx + 1$, the number of bits that differ from idx is the total of trailing bits that propagate a carry through the v trailing ones flipping each from 1 to 0. Therefore, exactly $v + 1$ bits change between idx and $idx + 1$: v bits from 0 through $v - 1$ that flip from 1 to 0 and 1 bit that flips from 0 to 1.

We now prove by induction on v that DeltaDecode correctly reconstructs $\mathcal{A}_{idx+1}$ at every level.

Base Case. ($v = 0$). $\nu(idx) = 0$ means idx is even, so, based on the construction of a Merkle Tree where the first index of the first leaf is 0 by the notation [National Institute of Standards and Technology 2024], the neighbor $idx + 1$ is odd. Exactly one level differs: level 0. DeltaEncode computes

$$\Delta_{idx}[0] \leftarrow auth[0]$$

and sets $\Delta_{idx}[0] = n_{0,k_0 \oplus 1}$ that represents the sibling of the leaf $idx + 1$ at level 0.

Induction Hypothesis. Assume that the lemma holds for all indices with $\nu(idx) = v - 1$.

Induction. Let $\nu(idx) = v$. By the definition of consecutive paths, levels 0 through v differ from $\mathcal{A}_{idx}$ to $\mathcal{A}_{idx+1}$ and levels $v+1$ through $h'-1$ stay the same. By the induction hypothesis, levels 0 through $v-1$ are correctly reconstructed. It remains to show level v, where DeltaEncode computes

$$\Delta_{idx}[v] \leftarrow auth[v]$$

and creates the correct node $n_{v,\,k_v \oplus 1}$ at level v. □

Transmission Cost. The transmission cost of Δ_{idx} is $(\nu(idx)+1) \cdot n$ bytes, compared to $h' \cdot n$ bytes, to transmit the full authentication path. In the worst case, where $idx = 2^{h'-1}-1$, $\nu(idx) = h'-1$ and delta equals the full authentication path.

5. Analysis

To evaluate Delta-XMSS we compare two metrics: (1) total number of bytes transmitted and (2) the computational overhead included by the delta construction. All experiments use $n = 32$ bytes based on the SHA-256 parameters set by RFC-8391 [Hülsing et al. 2018].

Security Remarks. The cached $\mathcal{A}_{idx}$ raises possible security concerns. An authentication path is a signature component to compute the tree from the leaf to the root. Its presence in the public signature means that it does not represent any risk under the EUF-CMA security model [Goldwasser et al. 1988]. Trivially, Delta-XMSS preserves the security of XMSS since the delta encoding affects only transmission, leaving the signature scheme unchanged. An interception and alteration of an authentication path would lead the receiver to reconstruct an incorrect tree root; since it would no longer match the trusted Master Public Key, the signature could not be verified.

5.1. Transmission Overhead

In standard XMSS every signing operation transmits a full authentication path of $h' \cdot n$ bytes. In Delta-XMSS only $(\nu(idx)+1) \cdot n$ bytes are transmitted, where $\nu(idx)$ is the number of trailing ones of idx. Cases where $\nu = 0$, which account for half of all signing indices, reduce the authentication path size by 90%. Table 1 compares the number of bytes transmitted comparing XMSS and Delta-XMSS and the additional overhead and CPU cycles caused by the implementation of Enc (DeltaEncode) and Dec (DeltaDecode).

Table 1. Delta-XMSS benchmark results ($h' = 10$, $n = 32$ bytes, XMSS-SHA2_10_256). Timings averaged over 10,000 repetitions per index.

idx	ν	XMSS (B)	Delta (B)	Enc (μs)	Dec (μs)	Enc / Dec (cycles)
0	0	320	32	0.0026	0.0048	8 / 15
1	1	320	64	0.0040	0.0071	12 / 22
3	2	320	96	0.0052	0.0099	16 / 31
7	3	320	128	0.0117	0.0154	37 / 49
15	4	320	160	0.0152	0.0179	47 / 55
31	5	320	192	0.0132	0.0213	44 / 72

Relative to the complete signature, the reduction is bounded by the share of the authentication path. For XMSS-SHA2_10_256 the complete signature has 2,500 bytes [Hülsing et al. 2018]: a 4-byte index, a 32-byte randomization, a 2,144-byte WOTS+ signature and a 320-byte authentication path. So, the path accounts for 12.8% of the signature. Delta-XMSS reduces the total signature size by 11.52% when $\nu = 0$ and by 10.24% in the average case ($\mathbb{E}[\nu] \approx 1$), which is the best reduction by any delta technique, since these are the only bytes of the signature that can repeat across the signatures. Since the WOTS+ size does not depend on h', the relative gain grows with tree height, reaching 20.4% on average for XMSS-SHA2_20_256.

Timing measurements present a higher variance at higher indices due to system-level noise of operations at a sub-microsecond level. The results confirm that the additional computational overhead of Delta-XMSS is negligible, relative to the XMSS signing operation itself: 80 CPU cycles for the most frequent indices $\nu \leq 5$ compared to a median of 5,711,700. All experiments were conducted on an Intel i5-12500H running Ubuntu 22.04 (WSL2, x86-64), compiled with GCC 13.3. The implementation, written in C(99), is available at `https://github.com/antonioatra/xmss-reference`, including the benchmark scripts used in this evaluation.

6. Limitations

Our solution assumes strictly sequential signing (idx $\rightarrow$ idx+1). If signatures are skipped or a delta is lost during transmission, the receiver becomes desynchronized and cannot reconstruct subsequent authentication paths. Additionally, the first signature (idx = 0) requires transmitting the full authentication path, as no prior state exists in cache.

Delta-XMSS also trades receiver-side storage for bandwidth. The sender needs no additional state, since DeltaEncode operates only on the freshly generated SM_{idx+1}. The receiver, stateless in standard XMSS verification, must retain the last authentication path ($h' \cdot n$ bytes, or 320 bytes for our parameters) throughout the interval between two consecutive signatures. This favors bandwidth-constrained settings, but in sparse-signing scenarios such as occasional firmware updates, an average saving of 256 bytes per signature may not justify persistent receiver state.

7. Conclusion and Future Work

We proposed Delta-XMSS, a delta encoding scheme for XMSS authentication paths that reduces its transmission overhead on consecutive signing operations. By transmitting only $(\nu(idx) + 1) \cdot n$ bytes of the nodes that differ from $\mathcal{A}_{idx}$ and $\mathcal{A}_{idx+1}$, we can get a significant reduction in transmission cost.

As a future work, we want to introduce keyframes as a method to mitigate the local loss of information about the authentication path stored in cache and the delta encoding transmitted, and to visualize the impact of Delta-XMSS on IoT systems.

References

Buchmann, J., Dahmen, E., and Hülsing, A. (2011). XMSS – a practical forward secure signature scheme based on minimal security assumptions. In *Post-Quantum Cryptography (PQCrypto 2011)*, volume 7071 of *Lecture Notes in Computer*

Science, pages 117–129. Springer. Available: https://doi.org/10.1007/978-3-642-25405-5_8.

Buchmann, J., Dahmen, E., and Schneider, M. (2008). Merkle tree traversal revisited. In *Post-Quantum Cryptography (PQCrypto 2008)*, volume 5299 of *Lecture Notes in Computer Science*, pages 63–78. Springer. Available: https://doi.org/10.1007/978-3-540-88403-3_5.

Goldwasser, S., Micali, S., and Rivest, R. L. (1988). A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308. Available: https://doi.org/10.1137/0217017.

Hülsing, A., Butin, D., Gazdag, S., Rijneveld, J., and Mohaisen, A. (2018). XMSS: eXtended Merkle Signature Scheme. RFC 8391, IETF. Available: https://doi.org/10.17487/RFC8391.

Hülsing, A. and Rijneveld, J. (2018). XMSS reference implementation. Available: https://github.com/XMSS/xmss-reference.

Hunt, J. W. and McIlroy, M. D. (1976). An algorithm for differential file comparison. Technical Report Computing Science Technical Report 41, Bell Laboratories.

Jindal, R., Kumar, N., and Patidar, S. (2022). IoT streamed data handling model using delta encoding. *International Journal of Communication Systems*, 35(13). Available: https://doi.org/10.1002/dac.5243.

Jobst, M., Liu, C., Partzsch, J., Yan, Y., Kappel, D., Gonzalez, H. A., Ji, Y., Vogginger, B., and Mayr, C. (2020). Event-based neural network for ecg classification with delta encoding and early stopping. In *6th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–4. IEEE. Available: https://doi.org/10.1109/EBCCSP51266.2020.9291357.

Merkle, R. C. (1990). A certified digital signature. In *Advances in Cryptology – CRYPTO 1989*, volume 435, pages 218–238. Springer. Available: https://doi.org/10.1007/0-387-34805-0_21.

National Institute of Standards and Technology (2020). Recommendation for stateful hash-based signature schemes. Technical Report SP 800-208, NIST. Available: https://doi.org/10.6028/NIST.SP.800-208.

National Institute of Standards and Technology (2024). Stateless hash-based digital signature standard (SLH-DSA). Technical Report FIPS 205, NIST. Available: https://doi.org/10.6028/NIST.FIPS.205.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126. Available: https://doi.org/10.1145/359340.359342.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 124–134. IEEE. Available: https://doi.org/10.1109/SFCS.1994.365700.