

# rANS Signature Compression Done Right

HuFu Team

September 2023

## Abstract

As a compact and efficient entropy coding, the range variant of asymmetric numeral systems (rANS) has gained its attention for compressing the large lattice-based signatures. Among 40 NIST additional PQC digital signature candidates, there are two candidates, HuFu and HAETAE, that employ rANS to reduce their signature size. However, both of their implementations have flaws that result in severe signature forgery or buffer overflow under two types of attacks: the bit-flipping attack and the length modification attack. In this short report, we analyse the cause of these attacks and show that the existing rANS implementations can not fulfill the security requirements of signature compression. To address this issue, we propose a paradigm to safeguard the signature compression with minimal modification to rANS, which is also applicable to other asymmetric numeral systems such as tANS. This strategy will be applied on an updated version of HuFu to achieve a near-optimal signature size without sacrificing security.

## 1 Introduction

The range variant of asymmetric numeral systems (rANS) [1] is an entropy coding that compresses data almost to its entropy limit with high performance. As proposed by [2], lattice-based signatures that composed of large low-entropy vectors can benefit from rANS coding to significantly reduce their size. Compared to Falcon’s unary coding [3], the rANS coding can achieve a 7%-14% space improvement for compressing Gaussian signatures. Two of forty NIST additional PQC digital signature candidates, HuFu [4] and HAETAE [5], have adopted this technique to gain a shorter signature, and FuLeeca [6] has listed it as a future work. However, Saarinen [7, 8] recently demonstrated two attacks on the rANS implementations of HuFu and HAETAE, namely the bit-flipping attack and the length modification attack, which can result in signature forgery or buffer overflow. In this report, we will identify the causes of these vulnerabilities, and propose a framework to securely apply rANS for signature compression.

## 2 Safeguard the rANS Signature Compression

### 2.1 Defense the Bit-flipping Attack

Both HuFu and HAETAE are digital signature schemes that guarantee strong unforgeability (SUF-CMA), which means that an adversary cannot produce a valid signature pair  $(m^*, \sigma^*)$  that differs from any previously seen signature pairs  $(m_i, \sigma_i)$ . However, Saarinen discovered that an adversary can flip some bits in HuFu’s signature and generate a new valid signature for the same message [7]. He attributes this bit-flipping attack to an ambiguity of HuFu’s signature data structure, but this is actually a general attack affecting all rANS (and other asymmetric numeral systems) encoded signatures, including HAETAE: By altering the rANS coding of the highbits of  $[z_1]$ , an adversary can generate a new valid signature for HAETAE. For instance, changing byte 1318 in the first test vector of HAETAE-Level2 from “...BFA81C51A3CF7B...” to “...BFA81C519ECF7B...” would create a forgery.

The root cause of this attack lies in the fact that the asymmetric numeral systems allow the encoders to start from any initial states [9]. As a result, the same message can have multiple codings, and some of them

---

**Algorithm 1:** rANSEncode

---

**Parameter:** the radix of numeral system  $b$ , the state interval  $I$ , the initial state for the encoding  $x_{init} \in I$ , the width of the packed state  $w$ , the state intervals of symbols  $I_{\text{sym}}$   
**Input:** the symbols to encode ( $\text{sym}_1, \dots, \text{sym}_n$ )  
**Output:** a rANS coding ( $\text{str}_{j-w+1}, \dots, \text{str}_j, \text{str}_{j+1}, \dots, \text{str}_{-1}$ )

```
1:  $x \leftarrow x_{init}$  ▷ Fix the initial state for encoding
2:  $j \leftarrow -1$ 
3: for  $i \leftarrow n$  to 1 do ▷ Encode symbols backwards
4:   while  $x \notin I_{\text{sym}_i}$  do ▷ normalize the state
5:      $\text{str}_j \leftarrow x \bmod b$ 
6:      $j \leftarrow j - 1$  ▷ Emit data backwards
7:      $x \leftarrow \lfloor x/b \rfloor$ 
8:   end while
9:    $x \leftarrow C(\text{sym}_i, x)$  ▷ Encode a symbol
10: end for
11:  $(\text{str}_{j-w+1}, \dots, \text{str}_j) \leftarrow \text{PackState}(x)$ 
12: return  $(\text{str}_{j-w+1}, \dots, \text{str}_j, \text{str}_{j+1}, \dots, \text{str}_{-1})$ 
```

---

only differ by a few bits. The non-uniqueness of coding has no impact on general data compression tasks, but it can be a critical disadvantage for signature compression, as an adversary can easily forge different signatures by selecting a different coding.

To address this issue, we can exploit the unique decoding property of rANS, which shows that for any given symbol and state, there is only one previous state that can be decoded to them [1, 10]. This implies that, given a fixed final state, there is only one rANS coding that can be decoded to a specific message. Therefore, to ensure each message has a unique rANS coding, we can choose a constant number as the initial state of the encoder, and only accept the decoded message if the final state of the decoder matches that number. This technique can also be applied to safeguard other asymmetric numeral systems that have the unique decoding property, such as tANS [1]. The cost of fixing the initial state of encoding is only a few negligible bits, and it can be further reduced by storing a fixed number of symbols in its least significant bits [1].

Unfortunately, although most of the rANS and tANS implementations [11–13] have already fixed the initial state for the encoding as shown in algorithm 1, they do not verify the final state of the decoding, including the one used by HuFu and HAETAE [14]. Therefore, we modify these decoders to check if their final state matches the encoder’s initial state. We also verify that the decoder’s initial state is within a valid range and the input coding is entirely used up. The modifications are marked as blue in algorithm 2. This guarantees that the messages and codings are bijective, so forging a compressed signature is as hard as forging an uncompressed one. These modifications have minimal impact on the performance, and can be easily integrated into the existing rANS and tANS implementations.

## 2.2 Defense the Length Modification Attack

Since rANS is a variable length coding, the compressed data has different lengths for different messages. Therefore, both HuFu and HAETAE add fields to their signatures to indicate the length of the compressed signatures. However, this approach has a security risk, as an adversary can tamper with the length field to trigger buffer overflows [7, 8]. Instead of carefully checking this field, we can eliminate it by padding the compressed signature to a fixed length  $L$ . This way, we can ensure that the parsed length of the compressed message is always consistent with the actual length. For most impletations that always emits full bytes and fills the buffer backwards, we could use byte padding  $0x00^{L-|\text{str}|-1} \parallel 0x80 \parallel \text{str}$ , which resembles ISO/IEC 7816-4 [15].

---

**Algorithm 2:** rANSDecode

---

**Parameter:** the radix of numeral system  $b$ , the state interval  $I$ , the initial state for the encoding  $x_{init} \in I$ , the width of the packed state  $w$ , the number of symbols  $n$

**Input:** a rANS coding  $(\text{str}_1, \dots, \text{str}_N)$

**Output:** the decoded symbols  $(\text{sym}_1, \dots, \text{sym}_n)$  or  $\perp$

```
1: if  $w > N$  then return  $\perp$  ▷ The coding is too short
2:  $x \leftarrow \text{UnpackState}(\text{str}_1, \dots, \text{str}_w)$  ▷ Read the initial state for decoding
3:  $j \leftarrow w + 1$ 
4: if  $x \notin I$  then return  $\perp$  ▷ The initial state is out of range
5: for  $i \leftarrow 1$  to  $n$  do
6:    $(\text{sym}_i, x) \leftarrow D(x)$  ▷ Decode a symbol
7:   while  $x \notin I$  do ▷ normalize the state
8:     if  $j > N$  then return  $\perp$  ▷ The coding is too short
9:      $x \leftarrow xb + \text{str}_j$ 
10:     $j \leftarrow j + 1$ 
11:   end while
12: end for
13: if  $x \neq x_{init}$  then return  $\perp$  ▷ The final state for decoding does not match the initial state for encoding
14: if  $j \neq N + 1$  then return  $\perp$  ▷ The coding has redundant parts
15: return  $(\text{sym}_1, \dots, \text{sym}_n)$ 
```

---

### 3 Conclusion

This report examines two types of attacks on the rANS coding of HuFu and HAETAE, which are the bit-flipping attack and the length modification attack. The analysis reveals a practical gap between the existing rANS and tANS implementations and the security requirements of signature compression. To fill this gap, we propose two countermeasures: checking the final state of decoding with the initial state of encoding to prevent the bit-flipping attack, and padding the compressed signature properly to prevent the length modification attack. These countermeasures can securely compress the signatures with minimal modification to rANS, and are also suitable for other asymmetric numeral systems such as tANS. These techniques will be adopted in an updated version of HuFu to achieve a practical signature scheme that enjoys both compactness and security.

### References

- [1] Jarek Duda. Asymmetric numeral systems as close to capacity low state entropy coders. *CoRR*, abs/1311.2540, 2013.
- [2] Thomas Espitau, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Shorter hash-and-sign lattice-based signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 245–275, Cham, 2022. Springer Nature Switzerland.
- [3] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Submission to the NIST’s post-quantum cryptography standardization process, 2022. <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- [4] Yang Yu, Huiwen Jia, Leibo Li, Delong Ran, Zhiyuan Qiu, Shiduo Zhang, Xiuhua Lin, and Xiaoyun Wang. Hufu: Hash-and-sign signatures from powerful gadgets. <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/HuFu-spec-web.pdf>, 2023. Accessed: 31-8-2023.

- [5] Jung Hee Cheon, Hyeongmin Choe, Julien Devevey, Tim Guneysu, Dongyeon Hong, Markus Krausz, Georg Land, Junbum Shin, Damien Stehle, and MinJune Yi. Haetae: Hash-and-sign signatures from powerful gadgets. <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/haetae-spec-web.pdf>, 2023. Accessed: 31-8-2023.
- [6] Stefan Ritterhoff, Sebastian Bitzer, Patrick Karl, Georg Maringer, Thomas Schamberger, Jonas Schupp, Georg Sigl, Antonia Wachter-Zeh, and Violetta Weger. Fuleeca: Algorithm specifications and supporting documentation. <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/FuLeeca-spec-web.pdf>, 2023. Accessed: 31-8-2023.
- [7] Markku-Juhani O. Saarinen. Hufu: Big-flipping forgeries and buffer overflows. <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Hq-wRFDbIaU>, 2023. Accessed: 31-8-2023.
- [8] Markku-Juhani O. Saarinen. Buffer overflows in haetae / on crypto vs implementation errors. <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/ImcSqGLFdoo>, 2023. Accessed: 31-8-2023.
- [9] Jarek Duda. Asymmetric numeral systems. *CoRR*, abs/0902.0271, 2009.
- [10] Fabian Giesen. rans notes. <https://fgiesen.wordpress.com/2014/02/02/rans-notes/>, 2014. Accessed: 2023-08-17.
- [11] James Bonfield. jkbonfield/rans\_static. [https://github.com/jkbonfield/rans\\_static](https://github.com/jkbonfield/rans_static), 2022. Accessed: 31-8-2023.
- [12] Jeff Johnson. Dietgpu: Gpu-based lossless compression for numerical data. <https://github.com/facebookresearch/dietgpu>, 2022. Accessed: 31-8-2023.
- [13] Yann Collet. Cyan4973/finitestatentropy. <https://github.com/Cyan4973/FiniteStateEntropy/>, 2022. Accessed: 31-8-2023.
- [14] Fabian Rygorous. rygorous/ryg\_rans. [https://github.com/rygorous/ryg\\_rans](https://github.com/rygorous/ryg_rans), 2018. Accessed: 31-8-2023.
- [15] International Organization for Standardization. Identification cards – integrated circuit cards – part 4: Organization, security and commands for interchange. Standard, International Organization for Standardization, Geneva, CH, 2020.