Cryptanalysis of the Wave Signature Scheme

Paulo S. L. M. Barreto[1] and Edoardo Persichetti[2]

[1] School of Engineering and Technology, University of Washington Tacoma
pbarreto@uw.edu
[2] Department of Mathematical Sciences, Florida Atlantic University
epersichetti@fau.edu

Abstract. In this paper, we cryptanalyze the signature scheme Wave, which has recently appeared as a preprint. First, we show that there is a severe information leakage occurring from honestly-generated signatures. Then, we illustrate how to exploit this leakage to retrieve an alternative private key, which enables efficiently forging signatures for arbitrary messages. Our attack on the proposed 128-bit secure Wave parameters runs in about 13 minutes, most of which are actually spent collecting genuine signatures. We also explain how our attack applies to generalized versions of the scheme which could potentially be achieved using generalized admissible $(U, U+V)$ codes and larger field characteristics. Finally, as a target for further cryptanalysis, we describe a variant of Wave that we call Tsunami, which appears to thwart our attacks while keeping the positive aspects of that scheme.

1 Introduction

Digital signatures schemes are, without a doubt, one of the most important cryptographic protocols in use today, addressing the issue of authentication of data. Digital signatures share many similarities with confidentiality operations like encryption, key agreement, and key encapsulation; in some cases (e.g. RSA), this means that it is possible to design a signature scheme very naturally.

However, this is not always true for all families of cryptosystems, with one of the best examples being code-based cryptography. Despite code-based schemes constituting one of the leading families of proposed quantum-resistant (*aka* post-quantum) cryptosystems, designing efficient code-based signature schemes is in fact a major challenge and, to date, still an open problem. The reason for this is that the traditional hash-and-sign approach used for instance by RSA and other signatures is severely hindered by the fact that the subset of decodable syndromes of a linear code is typically only an exceedingly minute part of the total vector space, and so picking a vector at random (via hashing) most likely will not produce a decodable syndrome. This forces the signing algorithm to be repeated a prohibitively large number of times, creating an obviously impractical process. The authors of CFS [1], the first proposed signature scheme following this approach, propose particular choices of parameters to alleviate this issue, but the number of attempts required on average is still quite high (e.g. around 9!

for an estimated security level of 2^{80}) and the resulting signing times are several orders of magnitude slower than schemes based on other primitives. Moreover, "CFS-friendly" codes have a very high rate, and this can lead to potential additional issues such as distinguishers [4].

Alternatively, one could think of obtaining a signature scheme using the Fiat-Shamir transform and an identification scheme: this approach works very well for lattices, for instance [5]. However, the Hamming metric that comes with linear codes (instead of the Euclidean metric used for lattices) has proved to be too weak to provide security [6], and therefore the only plausible method to obtain code-based zero-knowledge is to use multiple commitments, as in Stern's scheme [7]. The problem with this type of schemes, though, is that multiple commitments allow the attacker to cheat and be successful with non-trivial probability (2/3 in Stern's original proposal), and therefore the identification scheme needs to be repeated multiple times to guarantee acceptance. This results in a very large signature size. Despite attempts to reduce this cheating probability [8], it seems implausible to obtain a truly efficient scheme in this way.

The recent Wave scheme [3] would be a nice development in this scenario. The scheme features an arguably elegant design and comes with a formal proof of security in the sense of existential unforgeability under chosen-message attacks (EUF-CMA). If that scheme withstood cryptanalysis, it would constitute a real and most welcome breakthrough.

Our Contribution: In this work, we show that, unfortunately, the Wave scheme leaks a considerable amount of private information in genuinely generated signatures. The leaked information is enough to recover an equivalent private key from the public key and a number of collected valid signatures, enabling an attacker to forge signatures for arbitrary messages without knowing the actual private key. The number of legitimate signatures the attacker needs to gather is fairly small (a few thousands) and the equivalent key recovery runs very fast in practice. The most time-consuming stage by far is the generation of the collected legitimate signatures.

For signature schemes equipped with EUF-CMA proofs, this is unusual, but not unprecedented. The Courtois-Finiasz-Sendrier (CFS) scheme [1] was similarly proven existentially unforgeable by Dallot [2] under the assumption that the underlying code adopted for the scheme is indistinguishable from a uniformly random linear error-correcting code. As it turned out, the only codes known to be suitable for CFS are very high-rate Goppa codes, which have since been shown to be distinguishable from random [4]. In other words, there is nothing wrong with the proof *per se*; rather, the hardness assumptions themselves fail to hold.

The same unfortunate circumstance is found here, with a crucial difference: an attacker can actually forge Wave signatures for arbitrary messages, rather than merely a single signature for some valid but contrived, meaningless message, while CFS signatures are, to the best of our knowledge, still empirically secure (it is just that the security proof does not hold for the chosen codes). This behavior is not only predicted theoretically by our attack, it is supported by empirical

evidence in the form of a Magma implementation that shows the attack to be entirely practical.

Given the nature of the property we exploit to mount our attack, we complete our contribution by proposing a variant of WAVE that we call the TSUNAMI scheme, which appears to prevent information leakage and hence to thwart the attacks. We do not claim TSUNAMI to be impervious to attacks, even though it seems to withstand all our present attempts: rather, it is meant to offer a refined target for further cryptanalysis efforts, and hence to stimulate continued investigation on the possibilities (or lack thereof) of attaining code-based signatures.

The remainder of this paper is organized as follows. We start by recapitulating theoretical notions in Section 2. We summarize the WAVE digital signature scheme in Section 3. In Section 4 we show how the structure of WAVE parity-check matrices unavoidably leaks structural information about the private key in each generated signature. In Section 5 we turn the leaked information into a full-fledged attack that recovers an equivalent private key and enables forging signatures. In Section 6 we present the results obtained via a simple Magma script. Section 7 discussed generalizations of our attack to other potential families of underlying codes mentioned in the WAVE specification. We propose the TSUNAMI variant in Section 8. We conclude in Section 9.
For the sake of completeness, we have included our Magma script, in Appendix A, to better illustrate our attack against the proposed 128-bit security parameters. That script also illustrates the TSUNAMI variant, conditioned to a suitably named Boolean switch.

*** *Update* #1*:* Since the first version of this report was ready, the authors of WAVE have contacted us to point out that, in our description of WAVE and in the script we include in the Appendix, we omit the rejection sampling they adopt to make sure the distribution of coefficients in the signatures is weakly uniform (that is, that signatures are statistically close to uniform).

In response, we point out that our attack does not depend on, and in fact is not related at all, to that distribution. As we describe in Section 4, our distinguisher is not based on biases on the overall distribution of signature coefficient. Rather, it can be viewed as a *differential* distinguisher: it keeps track of individual differences between certain components, not their distributions. This differential bias is not removed when some candidate signatures are rejected during signing: it cannot be removed, since doing so would necessarily mean violating the Hamming weight constraint on the signatures.

Indeed, the script in the Appendix can be readily adapted to simply use signatures generated elsewhere. Only the final forgery would have to be updated accordingly, namely, a few candidate signatures would have to be computed from the recovered equivalent and tested for whatever rejection sampling criterion is adopted. The same expected amount of rejections are expected for forged signatures as they are for genuine ones (between 25% and 33% according to the authors of WAVE).

*** *Update #2:* The authors of WAVE have also kindly provided us with explicit formulas for the rejection sampling mechanism involved in the D_V decoder. That mechanism is now taken into account in our attack script.

Summarizing: out attack still works against WAVE implemented in as fully detailed a fashion as the available specification makes possible; it is not thwarted by the the rejection sampling in the D_V decoder (in fact, it appears to benefit from it; see Section 8.1).

2 Preliminaries

Notation: We denote with $\mathbf{0}$ the all-zero matrix and with $\mathbf{I}$ the identity matrix.

Let p be a prime number and let n an k be positive integers with $k < n$. The *Hamming weight* of a vector $x \in \mathbb{F}_p^n$ is defined as the number $\mathsf{wt}(x)$ of its nonzero components. An (n, r)-*linear code* $\mathcal{C}$ of length n, dimension k and co-dimension $r = n - k$ is a k-dimensional vector subspace of $\mathbb{F}_p^n$. The code is called *ternary* in the case $p = 3$ (and p-*ary* in general).

Every linear code is spanned by the rows of some matrix $G \in \mathbb{F}_p^{k \times n}$, called a *generator matrix* of $\mathcal{C}$. Since bases of vector spaces are not unique, every code admits multiple generator matrices, each corresponding to a particular choice of basis. Equivalently, a linear code can be described as the kernel of a matrix $H \in \mathbb{F}_p^{r \times n}$, called *parity-check matrix*, i.e. $\mathcal{C} = \{c \mid cH^T = \mathbf{0}\}$. The *codeword* $c \in \mathcal{C}$ corresponding to a vector $m \in \mathbb{F}_q^k$ is given by $c = mG$. The *syndrome* $s \in \mathbb{F}_p^r$ of a vector $e \in \mathbb{F}_p^n$ is the vector $s := eH^T$.

A $(U, U + V)$ *code* is a linear code that admits a parity-check matrix of form

$$H := \begin{bmatrix} H_U & \mathbf{0} \\ -H_V & H_V \end{bmatrix} \in \mathbb{F}_p^{r \times n},$$

for some $H_U \in \mathbb{F}_p^{r_U \times (n/2)}$, $H_V \in \mathbb{F}_p^{r_V \times (n/2)}$, and $r_U + r_V = r$. Defining $k_U := n/2 - r_U$ and $k_V := n/2 - r_V$, the code dimension can be written $k = k_U + k_V$.

3 The Wave signature scheme

The original WAVE signature scheme follows the hash-and-sign approach. This means that signatures are obtained by decoding syndromes, which are (randomized) hashes of the message to be signed, into error patterns of a certain Hamming weight. Traditionally in code-based cryptography, this Hamming weight is required to be very low (well below the Gilbert-Varshamov bound), so that the honest signer can make use of his trapdoor, i.e. the decoding algorithm associated to the code in use, while decoding is hard for an attacker. However, this is not the only instance in which decoding is hard: as the authors of WAVE point out, the problem is easy if the weight grows beyond the bound, but it becomes hard again when it becomes very high.

The setting that is actually proposed in WAVE is the latter (see [3, "Tweaking the Prange Decoder for Reaching Large Weights"]), where the error pattern has

very high Hamming weight. The reason is that the trapdoor enables solving dense decoding instances which are seemingly much harder than sparse instances when the code characteristic p is odd. The trapdoor itself consists of the particular class of codes chosen, a generalized version of $(U, U+V)$ codes called *Generalized Admissible* $(U, U+V)$ codes. These codes can be decoded using a small variant of the $(U, U+V)$ decoder, for which the range of relative weights that are easy to decode is wider than the generic "easy" range (hence the trapdoor).

The only concrete instantiation specified by the authors of WAVE is in characteristic $p = 3$ and for plain $(U, U+V)$ codes. We will simplify our exposition accordingly, although we will also discuss the general case later.

3.1 Key pair

The WAVE private key is a triple (S, H_{sk}, P), where $S \in \mathbb{F}_3^{r \times r}$ is a nonsingular matrix, $H_{sk} \in \mathbb{F}_3^{r \times n}$ is the parity-check matrix of a linear code of even length n, co-dimension r (and dimension $k = n - r$), and $P \in \mathbb{F}_3^{n \times n}$ is a permutation matrix (i.e. each row and each column have only a single component of unit value, and all other components are zero), where H_{sk} is a generator matrix of a $(U, U+V)$ code, i.e. it has the shape:

$$H_{sk} := \begin{bmatrix} H_U & \mathbf{0} \\ -H_V & H_V \end{bmatrix} \in \mathbb{F}_3^{r \times n},$$

with $H_U \in \mathbb{F}_3^{r_U \times (n/2)}$ and $H_V \in \mathbb{F}_3^{r_V \times (n/2)}$. The public key associated to it is $H_{pk} := SH_{sk}P$.

3.2 Signing and verifying

To sign a message m, the WAVE scheme hashes m together with a uniform random nonce z to a syndrome reprentative $s' \leftarrow \mathcal{H}(m, z)$, then uses the private trapdoor (see Section 3.4) to solve the equation $e' H_{pk}^T = s'$ for a vector e' of weight $\mathsf{wt}(e') = w$. The complete signature is the pair (e', z).

This is achieved by observing that $e' H_{pk}^T = s' \Leftrightarrow e'(SH_{sk}P)^T = s' \Leftrightarrow e'(P^T H_{sk}^T S^T) = s' \Leftrightarrow (e'P^T)H_{sk}^T = s'(S^T)^{-1} \Leftrightarrow eH_{sk}^T = s$ where $e := e'P^T$ with $\mathsf{wt}(e) = w$ and $s := s'(S^T)^{-1}$.

Signature verification proceeds by simply recomputing $s' \leftarrow \mathcal{H}(m, z)$, then checking that $e' H_{pk}^T = s'$ and that $\mathsf{wt}(e') = w$.

3.3 Parameters

Only one set of parameters is suggested in the original description of WAVE, namely, ternary codes of length $n = 5172$, dimension $k = 3908$ and target Hamming weight $w = 4980$, with $k = k_U + k_V$, $k_U = 2299$, $k_V = 1609$ (and hence $r = r_U + r_V = 1264$, $r_U = 287$, $r_V = 977$). The Laplace distribution used in the D_V decoder sets $\alpha = 0.545$ and $\sigma = 18.81$ which are claimed to be optimal for security.

Here we see how the dense setting outperforms the more intuitive, sparse one: in this configuration, a sparse setting could only expect to decode a syndrome to an error pattern of weight around 747, while the dense setting decoding to $w = \lfloor n - (2/3)r_U \rfloor = 4980$ leaves only 192 zero columns. This clearly justifies WAVE's adoption of the dense setting.

3.4 Using the trapdoor for signing

The structure of the private parity-check matrix allows to decode a syndrome $s := (s_U, s_V)$ with $s_U \in \mathbb{F}_3^{r_U}$, $s_V \in \mathbb{F}_3^{r_V}$, into an error pattern $e := (e_U, e_U + e_V)$ with $e_U, e_V \in \mathbb{F}_3^{n/2}$. If we initially disregard the Hamming weight of the solution, we just need to obtain arbitrary solutions of the two linear systems $e_V H_V^T = s_V$ and $e_U H_U^T = s_U$, as one can check that $e = (e_U, e_U + e_V)$ is an arbitrary solution of $e H_{sk}^T = s$.

Solving those two linear systems without any further requirement is easy: choose k_U (resp. k_V) components of the desired error pattern and set them to arbitrary values, then solve the resulting square linear system for the remaining r_U (resp. r_V) components. Prange's method does essentially this, but is tailored to look for a solution of specifically low or high Hamming weight: instead of setting the chosen components to random values, set them all to zero (to obtain a low-weight solution) or else set them all to random nonzero values (to obtain a high-weight solution).

As the authors point out, one can actually do much better to obtain high-weight solutions of the target linear system $e H_{sk}^T = s$. The technique consists of choosing e_V entirely at random from a certain distribution, and then computing e_U to satisfy certain criteria. Specifically, since we want $e_U + e_V$ to have as few zero entries as possible, and given that the code has characteristic 3, it is enough to make as many entries on e_U equal to the corresponding entries on e_V if the latter are nonzero (if $e_V[j] = \pm 1$, setting $e_U[j] \leftarrow e_V[j]$ not only ensures $e_U[j] \neq 0$, but also $e_U[j] + e_V[j] = -e_U[j] \neq 0$), or else a random nonzero value if those entries on e_V are zero (in which case $e_U[j] = \pm 1$ at random and $e_U[j] + e_V[j] = e_U[j] \neq 0$). Consequently, all but r_U entries on e_U and as many on $e_U + e_V$ can be forced to be nonzero. The remaining r_U entries on e_U are computed as solutions to the linear system $e_U H_U^T = s_U$ and hence their values are not coercible to nonzero values, nor are the values of the corresponding entries on $e_U + e_V$, for the same reason. Yet, since those $2r_U$ values are expected to be roughly uniformly distributed, only about 1/3 of them will turn out to be zero, and the expected weight of $(e_U, e_U + e_V)$ is thus about $n - (2/3)r_U$.

Rejection sampling The actual WAVE scheme does not, however, pick just any solution to either $e_V H_V^T = s_V$ or $e_U H_U^T = s_U$, but filters them through rejection sampling at both D_V and D_U decoders.

For the first equation, the D_V decoder (Algorithm 3 in [3]) samples a target weight t for the k_V chosen components of the desired error pattern from the Laplace distribution (with mean $(1 - \alpha)k_V$ for a certain parameter α and a

certain variance σ), and after the solution e_V has been computed, applies rejection sampling mechanism designed to ensure that e_V is indistinguishable from a uniformly random element from $\mathbb{F}_3^{n/2}$ conditioned to having a certain expected Hamming weight. Interestingly, the expected Hamming weight of the computed solutions before rejection sampling step is about $(1-\alpha)k_V + (2/3)r_V \approx 1383$ for the actual WAVE, but tends to be slightly (about 2%) higher after rejection sampling since the acceptance rate is observed to peak at Hamming weight 1411 instead, while at Hamming weight 1383 the acceptance rate is only about 0.9 times as high.

For the second equation, the D_U decoder (Algorithm 4 in [3]) performs a related but more involved rejection sampling step. It samples target weights j_1 and j_{-1} for the numbers of 1 and -1 entries on the k_V chosen components of the desired error pattern from a 2-dimensional Laplace distribution with parameters tailored for the actual numbers t_1 and t_{-1} of 1 and -1 entries on e_V (presumably with mean $\mu_1 := (t_1/(n/2))k_U$ and $\mu_{-1} := (t_{-1}/(n/2))k_U$, matching the expected amounts of 1 and -1 entries on k_U uniformly randomly chosen columns from e_V, though [3] simply states that "the mean and variance of that distribution is optimized according to the first step output weight" without further specifications), and after the solution e_U has been computed, it counts the numbers ℓ_1 and ℓ_{-1} of positions where e_V is respectively 1 or -1 but e_U does not match it, and applies rejection sampling with acceptance rate conditioned to the observed ℓ_1 and ℓ_{-1}.

The WAVE parameters are chosen in such a way that the fraction of candidate signatures discarded by either rejection sampling mechanism is small (11% for D_V and 19% for D_U).

4 Structural leakage

We now show that, unless e_U and e_V are computed according to a very specific criterion (that the WAVE setting does not follow), it turns out to be possible to accumulate statistics from observed genuine signatures and recover an equivalent signing key.

Initially, consider the distribution of entry values on e_U in a scenario where rejection sampling is disabled for the D_U decoder (the effect of enabling rejection sampling for D_U will be discussed later, in Section 8.1).

We know that k_U of the e_U entries have their values chosen from a certain distribution, rather than computed as solutions of a linear system. A fraction about $\gamma := ((1-\alpha)k_V + (2/3)r_V)/(n/2)$ out of those $e_U[j]$ chosen entries, namely, those corresponding to $e_V[j] \neq 0$, are set to be equal to $e_V[j]$, in which case $e_U[j] + e_V[j] = -e_U[j]$. The remaining fraction $1-\gamma$, corresponding to $e_V[j] = 0$, are set to $e_U[j] = \pm 1$ uniformly at random, in which case $e_U[j] + e_V[j] = e_U[j]$.

The values of the r_U computed entries of e_U are close to uniform, as are the corresponding entries of $e_U + e_V$. In that case, if neither $e_U[j]$ nor $e_U[j] + e_V[j]$ is zero, which happens a fraction about 4/9 of the time, they are either equal or opposite with equal probability, namely, 2/9 of the time each.

Overall, $\gamma(n/2 - r_U) + (2/9)r_U = \gamma n/2 - (\gamma - 2/9)r_U$ entries of e_U are nonzero and opposite to their corresponding entry on $e_U + e_V$, $(1-\gamma)(n/2 - r_U) + (2/9)r_U = (1-\gamma)n/2 - (7/9 - \gamma)r_U$ entries of e_U are nonzero and equal to their corresponding entry on $e_U + e_V$, and $(5/9)r_U$ entries of e_U are zero or correspond to a zero entry on $e_U + e_V$.

Now let $e := (e_U, e_U + e_V)$. Then, discarding the entries where either e_U or $e_U + e_V$ are zero, we see that

$$\Pr\{e[j] = -e[j + n/2]\} \approx \frac{\gamma n/2 - (\gamma - 2/9)r_U}{n/2 - (5/9)r_U}$$

and

$$\Pr\{e[j] = e[j + n/2]\} \approx \frac{(1-\gamma)n/2 - (7/9 - \gamma)r_U}{n/2 - (5/9)r_U}.$$

Therefore, after the column permutation is applied to e to get the signature e', one still expects

$$\Pr\{e'[j] = -e'[h]\} \approx \frac{\gamma n/2 - (\gamma - 2/9)r_U}{n/2 - (5/9)r_U}$$

and

$$\Pr\{e'[j] = e'[h]\} \approx \frac{(1-\gamma)n/2 - (7/9 - \gamma)r_U}{n/2 - (5/9)r_U}$$

whenever two columns j and h correspond to the same original pair and their values are both nonzero. For the actual WAVE parameters this yields $\Pr\{e'[j] = -e'[h]\} \approx 0.53313$ and $\Pr\{e'[j] = e'[h]\} \approx 0.46687$. If columns j and h are nonzero but do *not* correspond to one same original pair, then $\Pr\{e'[j] = e'[h]\} = 1/2$ and $\Pr\{e'[j] = -e'[h]\} = 1/2$, since their values are independent and essentially uniform.

Thus, observing a certain amount of genuine signatures, it is possible in principle to accumulate enough statistics to infer which pairs of columns on the public parity-check matrix are more likely to correspond to matching pairs of columns on the secret parity-check matrix. To that end, among those signatures where any target columns j and h are both nonzero, count how many times their values are equal and how many times they are opposite: if the ratio of the counts is sufficiently close to $0.53313/0.46687 \approx 8/7$, these columns are likely a matching pair in the unpermuted code; if the ratio is closer to 1 instead, they are likely unrelated.

Signed permutations: One might think, at first glance, that adopting a *signed* column permutation P might thwart the accumulation of statistics. However, since the permutation is fixed for all signatures (it is a feature of the key pair), the actual result would merely be swapping the unbalanced probabilities without affecting the ratio of the larger to the smaller.

For instance, for the actual WAVE] parameters this would mean that, if the ratio of the above counts is sufficiently close to either $\approx 8/7$ or $\approx 7/8$, they likely

correspond to a matching pair in the unpermuted code, otherwise they are likely unrelated.

For simplicity, then, we henceforth only focus on the case where P is an unsigned permutation.

4.1 On leakage in the presence of rejection sampling

It is easy to see that the notion of weak uniform decoder does not capture all possibilities of attack, regardless of the values of the rejection sampling vectors adopted for D_U.

Consider the $\textsc{InfoSetW}(H, e, j_1, j_{-1})$ function, which is defined as a black box (only the pre- and post-conditions of its interface are specified). We will define a compliant function $\textsc{InfoSetW}'(H, e, j_1, j_{-1})$ that introduces a positional bias in its output, while still ensuring the defining conditions that the returned I is an information set of $H^\perp$ such that the number of positions $i \in I$ for which $e[i] = 1$ is equal to j_1 and $e[i] = -1$ is equal to j_{-1}. Assume that $\textsc{InfoSetW}_R$ is another compliant function that simply picks any information set at random.

function $\textsc{InfoSetW}'(H, e, j_1, j_{-1})$:
  repeat
    if $e[n-1] = 1$:
      $I' \leftarrow \textsc{InfoSetW}_R(H[0...r-1, 0...n-2], e[0...n-2], j_1 - 1, j_{-1})$
    if $e[n-1] = -1$:
      $I' \leftarrow \textsc{InfoSetW}_R(H[0...r-1, 0...n-2], e[0...n-2], j_1, j_{-1} - 1)$
    if $e[n-1] = 0$:
      $I' \leftarrow \textsc{InfoSetW}_R(H[0...r-1, 0...n-2], e[0...n-2], j_1, j_{-1})$
    $I \leftarrow n-1 \cup I'$
  until I is an information set of $H^\perp$
return I

Notice that I' is an infoset of cardinality $(n-r)-1$ and does not contain column $n-1$. Also, the counts of 1 and -1 entries on I' are such that column $n-1$ completes the required j_1 and j_{-1} counts. Therefore, position $n-1$ is in I with overwhelming probability (failure only happens when $j_1 = 0$, or $j_{-1} = 0$, or the rank of H is less than r, and any of this events only occurs with negligible probability). All other elements of I are chosen in precisely the same way they would if column $n-1$ had been chosen to be part of I by chance, so those are unbiased.

Now consider how the vector $\mathbf{x}$ is sampled within the D_U decoder: since $\text{Supp}(x) = I$ and column $n-1$ is now in I, necessarily $\mathbf{x}[n-1] \neq 0$. Therefore, $e_U[n-1]$ is always nonzero, and matches $e_V[n-1]$ when that element is itself nonzero. Also, column $n-1$ does not influence the values of ℓ_1 and ℓ_{-1}, since they are defined only in terms of columns where e_V is nonzero but e_U does not match it. As a consequence, vector $e = (e_U, e_U + e_V)$ has a positional bias at column $n-1$. Even after applying the secret permutation to get the actual signature, one of the columns, specifically, column $e[\pi(n-1)]$, is never zero, and this can

be detected with overwhelming probability $1 - \epsilon$ by observing a collection of $\epsilon/(\log w - \log n)$ signatures.

In conclusion, the D_U rejection sampling mechanism, being oblivious to positional biases (the $\mathbf{r}_U$ matrix of acceptance rates depends on counts of coefficient values and counts of (un)matches between corresponding coefficient values, but not on their positions, and the filtering is based on the values of ℓ_1 and ℓ_{-1}), cannot and does not filter out signatures generated when InfoSetW$'$ substitutes for InfoSetW. This means that D_U is not weakly uniform as assumed, and the security of Wave is therefore not guaranteed by Theorem 1.

5 Forging signatures

We now show how to exploit the structural leakage just described to forge signatures. First, we gather the necessary statistics from the collected signatures. Then, we recover the structure which allows to build an alternative private-key. Finally, we use this alternative key to sign any message.

5.1 Gathering statistics

There is a simple and effective way to gather the required statistics to recover the essential code structure. Let C be the matrix whose rows consist of all t collected signatures, each being a vector from $\mathbb{F}_3$. Lift the entries of C to $\mathbb{Z}$ with zero-centered entries (i.e. view C as a matrix of integers in range $\{-1, 0, 1\}$ rather than $\{0, 1, 2\}$). Then one can distinguish between matching and unmatching columns by looking for the largest entry on each row of $C' := -C^T C$.

This works because, when two given columns i and j do not match in the private code, the product of the corresponding coefficients on any given signature is either zero (in which case it does not contribute to the value of C'_{ij}), or else is uniformly distributed between -1 and 1. Either way, C'_{ij} will be simply a sum of up to t values uniformly chosen from $\{-1, 1\}$, with an expected value of zero and a standard deviation of $\sqrt{t}$.

However, when the columns *do* match, C'_{ij} will be the difference between the number of opposite and equal coefficients that occur at those columns among all collected signatures. Thus the expected value of C'_{ij} in this case is not zero but around $(8/7 - 1)t = t/7$ and again a standard deviation of around $\sqrt{t}$, with discrepancies due to the noise introduced when those columns are computed rather than chosen and neither is zero.

Except for those discrepancies due to noise, this distinguisher is likely to identify a pair of matching columns as long as the observed values between matching and unmatching columns do not coincide. Ignoring the noise, coincidence happens within β standard deviations when $0 + \beta\sqrt{t} = t/7 - \beta\sqrt{t}$, that is, when $\beta = \sqrt{t}/14$. For a desired (un)likeliness of failure (which corresponds to the probability of an event at or outside β standard deviations), this yields the approximate number of signatures needed to ensure success in guessing correctly one matching column pair, namely, $t = (14\beta)^2$. Here we use the central

limit theorem to approximate the distribution of C'_{ij} with a normal distribution (which is reasonable since the signature coefficients are, to a high enough precision, independent and identically distributed).

For instance, $\beta = 4$ corresponds to a probability of failure of $1/15787$ per column pair, or roughly $1/6$ for all pairs when $n = 5172$. To attain this, at least about $t = (1\beta)^2 = 3136$ signatures must be collected. In practice more signatures are required to compensate for the aforementioned noise. We observe that, without rejection sampling of any kind, an average of 6500 collected signatures (with a standard deviation of about 500) lead to a successful break.

5.2 Recovering the essential key structure

Although the above process does not reveal the exact column permutation P, it allows to group the columns of the parity-check matrix in such a way that matching pairs are exactly at a distance of $n/2$ apart. This grouping corresponds to applying a partial column permutation Q that keeps the overall structure (though not the actual coefficients) of the lower part of the private parity-check matrix unchanged, namely:

$$H_{pk}Q = S(H_{sk}PQ)$$

where

$$H_{sk}PQ = \begin{bmatrix} A' & B' \\ -W' & W' \end{bmatrix}$$

for some $A', B' \in \mathbb{F}_3^{r_U \times (n/2)}$ and $W' \in \mathbb{F}_3^{r_V \times (n/2)}$.

Moreover, doing this allows to compute a different basis for the same code where the structure of the lower part of the parity-check matrix is publicly visible. Namely, it becomes possible to find a matrix R such that:

$$RH_{pk}Q = \begin{bmatrix} A & B \\ -W & W \end{bmatrix}.$$

The obvious technique to achieve this is to compute the echelon form of a matrix Z consisting of the sum of the left and right sides of $H_{pk}Q$, namely:

$$Z := H_{pk}Q \cdot \begin{bmatrix} \mathbf{I} \\ \mathbf{I} \end{bmatrix} = S \cdot \begin{bmatrix} A' & B' \\ -W' & W' \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I} \\ \mathbf{I} \end{bmatrix} = S \cdot \begin{bmatrix} A' + B' \\ \mathbf{0} \end{bmatrix},$$

since the rank of Z is no more than r_U. Thus, the unscrambling matrix R we look for is one such that:

$$R \cdot Z = \begin{bmatrix} Z' \\ \mathbf{0} \end{bmatrix}$$

for some Z'. Such a matrix is effectively computed via plain Gaussian elimination, and one can check that $H_{bk} := RH_{pk}Q$ has the desired form.

It remains to show how to make use of H_{bk} to forge Wave signatures. We will discuss this in the next section.

5.3 Completing the forgery

As we have seen, the permutation Q and the unscrambling matrix R, recovered through leaks from genuine signatures, reveal part of the private parity-check structure, yielding a parity-check matrix H_{bk} with the structure:

$$H_{bk} := RH_{pk}Q = \begin{bmatrix} A & B \\ -W & W \end{bmatrix}.$$

A crucial remark is that the permutation Q likewise restores the structure $e' = (e'_U, e'_U + e'_V)$ of valid signatures, even though the actual coefficients probably differ from those of the actual signature. This means that $e'[j] = \pm e'[j + n/2] \neq 0$ for all but $\lfloor n - (2/3)r_U \rfloor$ entries of e' when $e'_V[j] \neq 0$, that the ratio between equal and opposite values in such matching pairs is about 2, and that the coefficients of e'_V are essentially random (only the coefficients of e'_U have to be chosen according to the same guidelines as before).

Now consider the task of decoding a syndrome $s' := (s_U, s_V)$ into a w-dense error pattern e' under the parity check matrix H_{bk}, that is, solving $e'H_{bk}^T = s'$. We have:

$$\begin{aligned} e'H_{bk}^T &= \begin{bmatrix} e'_U & e'_U + e'_V \end{bmatrix} \cdot \begin{bmatrix} A^T & -W^T \\ B^T & W^T \end{bmatrix} \\ &= \begin{bmatrix} e'_U A^T + (e'_U + e'_V)B^T & -e'_U W^T + (e'_U + e'_V)W^T \end{bmatrix} \\ &= \begin{bmatrix} e'_U (A+B)^T + e'_V B^T & e'_V W^T \end{bmatrix} \\ &= \begin{bmatrix} s'_U & s'_V \end{bmatrix}. \end{aligned}$$

This can be carried out by finding a random solution to $e'_V W^T = s'_V$, then solving $e'_U(A + B)^T = s'_U - e'_V B^T$. To keep the Hamming weight high, simply follow the same strategy as for the original WAVE private structure: set $n/2 - r_U$ coefficients $e'_U[j] = e'_V[j]$ when $e'_V[j] \neq 0$ or to a random nonzero value when $e'_V[j] = 0$, then solve for the remaining r_U coefficients.

To forge signatures for any message (properly hashed to a syndrome s), notice that any valid signature satisfies $eH_{pk}^T = s \Leftrightarrow (eQ)(Q^T H_{pk}^T R^T) = sR^T \Leftrightarrow e'H_{bk}^T = s'$, with $e' := eQ$ and $s' := sR^T$. Thus we can forge a WAVE signature e under key H_{pk} for a given syndrome s by recovering H_{bk} as shown in the previous section, then solving $e'H_{bk}^T = s'$ for e' of the correct weight with $s' := sR^T$, and finally returning $e = e'Q^T$.

6 Measurements and results

We have implemented our attack via a simple Magma script, which we have included in the Appendix, and we tested it on WAVE parameters. The machine used is an iMac, with processor Intel Core i5 at 3.2 GHz and 16Gb of RAM. The most expensive steps consists in collecting the required number of signatures (set to an average of 6500), which, in our measurements, is about 800 seconds (approximately 13 minutes). After that, the time spent to recover the matrix

structure corresponding to the alternative private key, and to produce a forgery, is negligible.

Note that, in the script, we actually allow for the Hamming weight of genuine signatures to be within 5% of the target weight. This is by no means a restriction of the attack: it is merely a way to speedup the generation of legitimate WAVE signatures, a process which could otherwise be unnecessarily slow: while it is not hard to get signatures whose weight is ever so slightly off the target $w = 4980$, a legitimate user would have to spend quite some time to obtain a signature of weight exactly w, perhaps tenfold or even longer.

7 Generalizing the attack

As we have mentioned, the WAVE scheme is formally defined on top of generalized admissible $(U, U+V)$ codes over some finite field $\mathbb{F}_p$. The authors choose to restrict the scheme description to plain $(U, U+V)$ codes of characteristic 3, justifying their choice for the sake of simplicity, but pointing out that their construction and analysis can be generalized. Although they refrain from providing explicit details on the use of these codes, we briefly show that they would still be vulnerable to simple variants of our attack, while also pointing out the limitations of our technique. We first consider the adoption of ternary generalized admissible $(U, U+V)$ codes, then we discuss larger characteristics.

7.1 Attacking generalized admissible $(U, U+V)$ codes

The detailed structure of the private parity-check matrix for a generalized admissible $(U, U+V)$ code is the following:

$$\begin{bmatrix} H_U D_4 M & -H_U D_2 M \\ H_V D_3 M & -H_V D_1 M \end{bmatrix},$$

where D_1 through D_4 and M are diagonal matrices, with D_1, D_3 and M invertible, and $M := (D_1 D_4 - D_3 D_2)^{-1}$. In this description, plain $(U, U+V)$ codes correspond to the particular choice of matrices $D_1 = D_3 = D_4 = \mathbf{I}$ and $D_2 = \mathbf{0}$.

Since M and D_1 through D_4 are diagonal, they commute. Moreover, given the condition that D_1 and D_3 are both invertible, we can define $E := D_3 \cdot D_1^{-1}$ and rewrite the above matrix in the following form as:

$$\begin{bmatrix} (H_U D_4 M) \cdot (D_3 D_1^{-1})^{-1} \cdot (D_3 D_1^{-1}) & (-H_U D_2 M) \\ -(-H_V D_1 M) \cdot (D_3 D_1^{-1}) & (-H_V D_1 M) \end{bmatrix}$$

or, equivalently,

$$\begin{bmatrix} AE & B \\ -WE & W \end{bmatrix} = \begin{bmatrix} A & B \\ -W & W \end{bmatrix} \begin{bmatrix} E & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

where $A := (H_U D_4 M) \cdot (D_3 D_1^{-1})^{-1}$, $B := -H_U D_2 M$ and $W := -H_V D_1 M$.

This form is very closely related to the recovered matrix in the attack against plain $(U, U+V)$ codes, the difference consisting only of the diagonal matrix:

$$F := \begin{bmatrix} E & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

on the right. The effect of multiplying an error pattern (without the private column permutation) by F is just to consistently flip the sign of certain paired columns. Indeed, this is the result of applying a specific signed permutation, which does not thwart the attack as seen in Section 4: for matching columns h and j, the ratio between the counts of occurrences with $e'[h] = -e'[j] \neq 0$ and those with $e'[h] = e'[j] \neq 0$ is either $\approx 8/7$ (when the diagonal element on F is 1) or $\approx 7/8$ (when that element is -1), while for unrelated columns the ratio is close to $1/2$. This enables finding matching columns and also the corresponding signal.

In conclusion, an attacker can still recover an equivalent private key for generalized admissible $(U, U+V)$ codes with only a small modification to the basic attack strategy for plain $(U, U+V)$ codes. The recovered equivalent trapdoor is even simpler than the structure of generalized admissible $(U, U+V)$ codes, since it does not require that complicated decomposition.

7.2 Larger characteristics

Assume now that the codes are defined over $\mathbb{F}_p$ for $p \geq 3$. The distinguisher we employ for ternary codes to pair up matching columns on legitimate signatures works by matching pairs (h, j) of columns that are not selected to be the solutions of linear systems. It is based on the observation that $e'[j] = e'[h]$ only when $e_V[j] = 0$, while if the columns do not match, equality happens by chance among all nonzero values. Thus, we count the number of times the values of two columns match, and infer they match when the count is suitably different from $1/2$.

For $p = 3$ the distinguisher works with a comfortable margin (a fraction 0.46687 of equalities for matching columns against $1/2$ for unmatching ones). For larger characteristics all we need to make is a small modification to the way we count equalities/inequalities.

Assume for simplicity that e_V is sampled with as few zero entries as possible (a similar reasoning with more complicated formulas holds for unbalanced distributions). The generalized distinguisher for any $p \geq 3$ is designed to lead to an expected zero-sum when the columns do not match, but to an expected sum value of roughly t/p for matched columns, where t is the number of collected signatures. To attain this, we simply compute a *weighted* sum instead, assigning weight $-\frac{p-1}{2}$ to count occurrences of $e'[j] = e'[h]$ and weight $\frac{p-1}{2(p-2)}$ to count occurrences of $e'[j] \neq e'[h]$.

Thus, when the columns are unmatched, the fraction of equal coefficients between columns h and j is mapped from $\frac{1}{p-1}$ to $-\frac{1}{p-1} \cdot \frac{p-1}{2} = -\frac{1}{2}$, and the fraction of different coefficients on those columns is mapped from $\frac{p-2}{p-1}$ to $\frac{p-2}{p-1} \cdot \frac{p-1}{2(p-2)} = \frac{1}{2}$, yielding the expected zero-sum. In contrast, for matched columns

the fraction of equal coefficients is mapped from $\frac{1}{p}$ to $-\frac{1}{p} \cdot \frac{p-1}{2} = -\frac{p-1}{2p(p-2)} \cdot (p-2)$, and the fraction of different coefficients is mapped from $\frac{p-1}{p}$ to $\frac{p-1}{p} \cdot \frac{p-1}{2(p-2)} = \frac{p-1}{2p(p-2)} \cdot (p-1)$, therefore with a bias of $\frac{p-1}{2p(p-2)} \approx \frac{1}{2p}$ above the zero-sum in favor of matching column pairs.

Adapting the statistical analysis from Section 5.1 accordingly, the number of required signatures for this generalized distinguisher is $t \approx (2p\beta)^2$ where β is again the number of standard deviations that an event with the same probability as an attack failure would be away from the mean. Thus, the odds of attack success become negligible when $p^2 \in O(2^\lambda)$ for λ-bit security. More precisely, since the signature size is proportional to $\lg p$, the total number of collected bits from t signatures becomes $4\beta^2 p^2 \lg p$, and setting this value to 2^λ yields a refined lower bound for p, namely, $p \approx 2^{\lambda/2-1}/(\alpha\sqrt{\lambda})$. For instance, for $\lambda = 128$ and $\beta = 6$ (corresponding to a probability of roughly $n/506797346 \approx 2^{-17}$ that the attack fails for the proposed $n = 5172$), the minimum characteristic to thwart the attack would be $p \approx 2^{57}$.

Yet, it is far less clear how to choose secure parameters that prevent other types of vulnerability for p as large as this: it might well turn out that other vulnerabilities are introduced by this setting, which we therefore neither recommend nor claim to be secure at all. Investigating the possibilities left, if any, transcends the scope of this paper.

8 The Tsunami variant

The Wave signature scheme is conceptually elegant, sports a detailed concrete assessment of the coding-theoretical problems related to the hardness assumption, and comes with a formal security proof which still formally holds, even though the hardness assumption itself does not. One can therefore legitimately ask if there is any way of fixing its security shortcomings. We now present intriguing evidence that this just might be the case. We focus on characteristic $p = 3$ to keep as close as possible to the original Wave setting, but similar observations hold for higher characteristics as well.

The distinguisher we exploit to mount our attack is based on the property that, for matching columns h and j that happen not to have been chosen for the information set needed for signing, equality of signature coefficients $e'[h] = e'[j]$ holds a fraction 0.46687 of the time (specifically, when $e_V[j] = 0$ and hence $e_U[j] + e_V[j] = e_U[j]$), while inequality $e'[h] \neq e'[j]$ holds the remaining 0.53313 fraction (that is, when $e_V[j] \neq 0$ and hence $e_U[j] = e_V[j]$, leading to $e_U[j] + e_V[j] = -e_U[j]$). This distinguisher stems from the setting $\alpha = 0.545$ for the Laplace distribution in the D_V decoder.

We now propose the Tsunami variant, that coincides with Wave *except in the computation of* e_V (and possibly in the way rejection sampling is set, depending on its parameters). Specifically, we set the distribution of the e_V coefficients in such a way that equality and inequality of entries on the final signature are *equiprobable*, that is, $\Pr\{e'[h] = e'[j]\} = \Pr\{e'[h] \neq e'[j]\}$ for all columns h, j.

In other words, let ζ be the fraction of zero entries on the columns of e_V in the information set chosen for solving the linear system. The number of zero coefficients in that part is thus ζk_V, and the number of ± 1 coefficients there is $(1-\zeta)k_V$. Outside the information set, the expected number of zero entries is $\approx \frac{1}{3} r_V$ and the number of nonzero entries is $\approx \frac{2}{3} r_V$, since those entries are essentially uniform. We want the *total* number of zero entries on e_V to match the number of nonzero entries, since this nullifies the distinguisher. That is, we need $\zeta k_V + \frac{1}{3} r_V = (1-\zeta)k_V + \frac{2}{3} r_V$, which means $\zeta = \frac{1}{2}\left(1 + \frac{1}{3}\frac{r_V}{k_V}\right)$.

Algorithmically, this means choosing the k_V free coefficients of e_V (that is, those in the information set used to solve the linear system) to zero with probability ζ, and uniformly to ± 1 with probability $(1-\zeta)$.

For WAVE parameters apart from α (and the corresponding variance), we have $\zeta \approx 0.6012$, hence we must set $\approx 60.12\%$ of the free entries of e_V, or ≈ 967 columns, to zero, and the remaining $\approx 39.88\%$ of them, or ≈ 642 columns, uniformly to ± 1.

We remark that the weight distribution of the signatures themselves remains extremely close to uniform: the only weight affected by the TSUNAMI setting is that of e_V, not that of $e = (e_U, e_U + e_V)$.

Also, we stress once again that we do not claim this variant to be secure: it withstands the core of our attack, but there may exist other distinguishers that enable pairing up the columns of H_{sk} (or that break the scheme in an entirely different fashion). Still, since TSUNAMI appears to share all positive properties of WAVE and was designed to fix its only drawback known so far, it is proposed explicitly as a target for further cryptanalysis, and as motivation for follow-up research on the possibility (or otherwise) of achieving code-based signatures.

8.1 The effect of rejection sampling

We observed earlier that the peak acceptance rate for the D_V rejection sampling occurs at a slightly higher weight than $\gamma n/2 \approx 1383$, namely, at weight 1411. This is even farther away from the TSUNAMI weight, which is simply $n/4 = 1293$ at which the acceptance rate is only about 5%. Now this has a curious effect: the attack actually *benefits* from it.

Indeed, without rejection sampling the average number of signatures needed for a successful attack is about 6500. Turning the D_V rejection sampling on, that number is observed empirically to slightly drop to an average of 6300 signatures.

Computing the acceptance rates for D_V and especially for D_U is no easy task: despite their apparent simplicity in the statement of [3, Theorem 1], they are only obtained for D_V by virtue of [3, Proposition 1], and are entirely missing from [3] for D_U.

However, it is quite unlikely that the signatures that pass that final layer of rejection sampling will have any substantial effect on the attack: thwarting it would require countering the effects of both the chosen α and that of the D_V rejection sampling, effectively filtering out the signatures where the weight of e_V was far away from $n/4$ and thus emulating a scenario where α is replaced by

ζ. Since only about 5% of those signatures survive the D_V rejection sampling itself, such a setting would decrease the acceptance rate at the D_U decoder from about 81% down to 5% or less, and even if it did, one would be left wondering why α was not set to ζ to begin with.

Nevertheless, in the absence of explicit formulas for the D_U we did a simple experiment and replaced 19% of the signatures by entirely random noise (so they do not even verify) rather than by different but valid signatures. The attack is still observed to work, the only effect being an increase in the number of necessary signatures to an average of 12000 and a standard deviation of 1000.

9 Conclusion

We have described distinguishers for (plain or generalized admissible) $(U, U+V)$ codes in small-to-moderate characteristic, exploiting leaks in genuinely generated signatures that enable recovering an equivalent signing key. The recovered key can in turn be used to sign arbitrary messages, not merely to produce an existential forgery. Thus the attack constitutes a total break of Wave, and shows that the hardness assumption required for that scheme to be secure does not hold.

The attack is practical as corroborated by empirical assessment (see the Appendix for a simple but complete implementation). Proposed parameters for the 128-bit classical security level can be broken in about a minute, with the actual key recovery taking only a few seconds after a modest number of genuine signatures are collected.

We have also proposed a potential fix in the form of the Tsunami variant. While we do not claim it to resist all kinds of structural attacks, it was designed specifically to withstand ours while keeping all positive aspects of Wave, and is thus offered as a target for further cryptanalysis. We hope this helps fostering continued research into the possibility (or otherwise) of achieving code-based digital signatures.

Acknowledgments

We are grateful to the authors of Wave for their comments on preliminary versions of this document.

References

1. Courtois, N.T., Finiasz, M., Sendrier, N.: How to achieve a McEliece-based digital signature scheme. In: Boyd, C. (ed.) Advances in Cryptology — ASIACRYPT 2001. pp. 157–174. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
2. Dallot, L.: Towards a concrete security proof of Courtois, Finiasz and Sendrier signature scheme. In: Lucks, S., Sadeghi, A.R., Wolf, C. (eds.) Research in Cryptology. pp. 65–77. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

3. Debris-Alazard, T., Sendrier, N., Tillich, J.P.: Wave: A new code-based signature scheme. arXiv preprint arXiv:1810.07554 (2018)
4. Faugère, J.C., Gauthier-Umaña, V., Otmani, A., Perret, L., Tilllich, J.P.: A distinguisher for high-rate McEliece cryptosystems. IEEE Transactions on Information Theory **59**(10), 6830–6844 (Oct 2013). https://doi.org/10.1109/TIT.2013.2272036
5. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) Advances in Cryptology – ASIACRYPT 2009. pp. 598–616. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
6. Persichetti, E.: Improving the efficiency of code-based cryptography. Ph.D. thesis, ResearchSpace@ Auckland (2012)
7. Stern, J.: A new identification scheme based on syndrome decoding. In: Stinson, D.R. (ed.) Advances in Cryptology — CRYPTO' 93. pp. 13–21. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
8. Véron, P.: Improved identification schemes based on error-correcting codes. Appl. Algebra Eng. Commun. Comput. **8**(1), 57–69 (1996)

A Magma script to illustrate the attack

The following script matches the available description of Wave except for the D_U rejection sampling layer, which is only implicitly defined in [3] (see the discussion in Section 8.1).

```
ZZ := Integers();
K := GF(3);

/*
 * Rejection sampling infrastructure for {\Wave}.
 *
 * Inspired by software kindly provided by T. Debris-Alazard, J.-P. Tillich, and N. Sendrier.
 */

function defN(kV, mu, sigma)
    return (1/2)*(&+[Exp((i+1-mu)/sigma) - Exp((i-mu)/sigma) : i in [0..Round(mu)-1]]
                + &+[Exp((mu-i)/sigma) - Exp((mu-i-1)/sigma) : i in [Round(mu)..kV]]);
end function;

function proba(mu, sigma, N, t)
    return (t lt mu)
        select (1/(2*N))*(Exp((t+1-mu)/sigma) - Exp((t-mu)/sigma))
        else   (1/(2*N))*(Exp((mu-t)/sigma) - Exp((mu-t-1)/sigma));
end function;

function q1(n, rV, i, probavec, binomvec, pow2vec)
    return &+[probavec[t + 1]*binomvec[(i-t) + 1]*pow2vec[(i-t) + 1] :
        t in [0..i]]/3^rV;
end function;

function q1u(n, w, i, pow2vec, binomvec, binomipvec)
    //assert n mod 2 eq 0;
```

```
    //assert w mod 2 eq 0;
    res := 0.0;
    for p := 0 to i by 2 do
        if (w+p) gt n then
            break; // all remaining terms are zero
        end if;
        if 2*i gt (w+p) then
            break; // all remaining terms are zero
        end if;
        res +:= 1.0*binomipvec[p + 1]*Binomial((w+p) div 2, i)
            *binomvec[(w+p) div 2 + 1]*pow2vec[(w+p) div 2 + p + 1];
    end for;
    return res/(Binomial(n, w)*2^w);
end function;

/**
 * Compute the acceptance rate vector needed for rejection sampling.
 * NB: choosing sigmaV < 1 disables rejection sampling
 * by setting all entries to 1.
 */
function MakeRsV(n, w, kV, alpha, sigmaV)
    n2 := n div 2; rV := n2 - kV; assert n mod 2 eq 0; assert kV lt n2;
    if sigmaV ge 1 then // actual rejection sampling (Wave)
        mu := (1 - alpha)*kV;
        N := defN(kV, mu, sigmaV);
        "N =", N;

        probavec := [proba(mu, sigmaV, N, j) : j in [0..n2]];
        binomvec := [Binomial(rV, j) : j in [0..n2]];
        pow2vec := [2^j : j in [0..n]];
        "computing q1vec...";
        time q1vec := [q1(n, rV, i, probavec, binomvec, pow2vec) :
            i in [0..n2]];
        "q1vec has been computed";

        binomvec := [Binomial(n2, j) : j in [0..(w + n2) div 2]];
        binomipvec := [1 : j in [0..n2]];
        "computing q1uvec...";
        q1uvec := [0.0 : i in [0..n2]];
        time for i in [0..n2] do
            q1uvec[i + 1] := q1u(n, w, i, pow2vec, binomvec, binomipvec);
            for p in [0..i] do
                binomipvec[p + 1] *:= (i + 1)/(i + 1 - p);
            end for;
        end for;
        "q1uvec has been computed";

        rsV := [q1uvec[i + 1]/q1vec[i + 1] : i in [0..n2]]; // raw vector
        M, wmax := Max(rsV); wmax -:= 1;
        invM := 1/M;
```

```
        "M =", M, ":", invM;
        "weight accepted with max prob:", wmax;
        rsV := [invM*rsV[i + 1] : i in [0..n2]];
    else // accept signatures without rejection sampling (Tsunami)
        rsV := [1 : i in [0..n2]];
    end if;
    return rsV; // acceptance rate vector
end function;

/*
 * General support for {\Wave} signatures and the {\Tsunami} variant.
 */

/**
 * Compute the Hamming weight of a vector.
 */
function Hweight(e)
    return &+[(e[k] ne 0) select ZZ!1 else ZZ!0 : k in [1..Ncols(e)]];
end function;

/**
 * Sample from the Laplace distribution of mean mu and variance sigma.
 */
function Laplace(mu, sigma)
    // sigma = 2*b^2
    b := Sqrt(sigma/2);
    // sample Z from Laplace(0, 1):
    X := 1.0*Random(0, 2^128 - 1)/2^128;
    Y := 1.0*Random(0, 2^128 - 1)/2^128;
    Z := Log(X/Y);
    // sample L from Laplace(mu, b):
    return Round(mu + b*Z);
end function;

/**
 * The D_V decoder.
 */
function D_V(sV, VT, alpha, sigmaV, rsV)
    n2 := Nrows(VT); rV := Ncols(VT); kV := n2 - rV;
    assert Ncols(sV) eq rV;
    A := {u : u in K} diff {0};
    mu := (1 - alpha)*kV;
    aatt := 0;
    repeat
        aatt +:= 1;
        // choose kV components:
        repeat
            eV := Vector(K, n2, [0 : j in [1..n2]]);
            ss := sV;
            // pick weight t in [0..kV] from the Laplace distribution:
```

```
            repeat
                t := Laplace(mu, sigmaV);
            until t ge 0 and t le kV;
            J := {1..n2}; // all available columns
            // choose t columns out of the kV components to guess uniformly to a nonzero value:
            for c in [1..t] do
                j := Random(J); J diff:= {j};
                // guess one component and update the syndrome accordingly:
                eV[j] := Random(A);
                ss -:= eV[j]*VT[j];
            end for;
            // choose kV - t components to be set to zero:
            for c in [1..kV - t] do
                j := Random(J); J diff:= {j};
                eV[j] := 0;
            end for;
            // group together the remaining coefficients
            // (to be computed as the solution to a linear system):
            L := [];
            for j in [1..n2] do
                if j in J then
                    L cat:= [j];
                end if;
            end for;
            HH := VerticalJoin([VT[L[i]] : i in [1..rV]]);
        until Determinant(HH) ne 0; // make sure the linear system admits a (unique) solution
        // compute the remaining coefficients:
        ee := ss*HH^-1;
        c := 0;
        for j in [1..n2] do
            if j in J then
                c +:= 1; eV[j] := ee[c];
            end if;
        end for;
        assert c eq rV;
        //assert eV*VT eq sV;
        // perform rejection sampling:
        weV := Hweight(eV);
        //"*** wt(e_V) =", weV, ":", 1.0*((1 - alpha)*kV + (2/3)*rV);
        acc := Random(0, 2^128-1) le rsV[weV + 1]*2^128; // accept with probability rsV[wt(eV)]
    until acc or aatt gt 100;
    //"***", aatt, ":", weV;
    assert aatt le 100;
    return eV;
end function;

/**
 * The D_U decoder.
 */
function D_U(sU, UT, eV, sigmaU, rsU)
```

```
n2 := Nrows(UT); rU := Ncols(UT); kU := n2 - rU;
assert Ncols(sU) eq rU; assert Ncols(eV) eq n2;
A := {u : u in K} diff {0};
// count the number of 1's and -1's on eV:
t1 := &+[(eV[j] eq  1) select ZZ!1 else ZZ!0 : j in [1..n2]];
tm := &+[(eV[j] eq -1) select ZZ!1 else ZZ!0 : j in [1..n2]];
t0 := n2 - t1 - tm;
//"========> t1 =", t1, ": tm =", tm, ": t0 =", t0, ": t1 + tm =", t1 + tm, ":", kU;
// select the mean values for the Laplace distributions
// (take them to match the expected amounts of 1's and -1's
// on kU randomly chosen columns from eV):
mu1 := (t1/n2)*kU; // NB: the Wave paper is unclear here; this is just a guess! <<<<<<<<<<<<<<<<<
mum := (tm/n2)*kU; // NB: the Wave paper is unclear here; this is just a guess! <<<<<<<<<<<<<<<<<
//"========> mu1 =", mu1, ": mum =", mum;
aatt := 0;
repeat
    aatt +:= 1;
    // choose kU components:
    repeat
        eU := Vector(K, n2, [0 : j in [1..n2]]);
        ss := sU;
        // pick weights j1, jm in [0..kV] from the Laplace distribution:
        repeat
            j1 := Laplace(mu1, sigmaU);
            jm := Laplace(mum, sigmaU);
        until j1 + jm le kU
            and j1 gt 0 and j1 le t1 // ensure there are at least j1 1-entries on eV
            and jm gt 0 and jm le tm // ensure there are at least jm (-1)-entries on eV
            and kU - j1 - jm le t0   // ensure there are at least (kU-j1-jm) 0-entries on eV
            ;
        //"*** j1 =", j1, ": jm =", jm, ": j1 + jm =", j1 + jm, ":", kU, ":", kU - j1 - jm;
        J := {1..n2}; // all available columns to pick the information set from
        // choose j1 columns to match the corresponding  1-entry on eV:
        J1 := {(eV[j] eq  1) select j else 0 : j in [1..n2]} diff {0}; // all 1-columns
        for c in [1..j1] do
            j := Random(J1); J1 diff:= {j}; J diff:= {j};
            // set one entry of eU to match the corresponding entry of eV,
            // and update the syndrome accordingly:
            eU[j] := eV[j]; assert eU[j] eq 1;
            ss -:= eU[j]*UT[j];
        end for;
        // choose jm columns to match the corresponding -1-entry on eV:
        Jm := {(eV[j] eq -1) select j else 0 : j in [1..n2]} diff {0}; // all (-1)-columns
        for c in [1..jm] do
            j := Random(Jm); Jm diff:= {j}; J diff:= {j};
            // set one entry of eU to match the corresponding entry of eV,
            // and update the syndrome accordingly:
            eU[j] := eV[j]; assert eU[j] eq -1;
            ss -:= eU[j]*UT[j];
        end for;
```

```
            J0 := {(eV[j] eq 0) select j else 0 : j in [1..n2]} diff {0}; // all 0-columns
            assert #J0 ge kU - j1 - jm;
            // choose kU - j1 - jm columns from those corresponding to a 0-entry on eV,
            // set them to uniformly random nonzero values, and update the syndrome accordingly:
            for c in [1..kU - j1 - jm] do
                j := Random(J0); J0 diff:= {j}; J diff:= {j};
                // guess one entry and update the syndrome accordingly:
                eU[j] := Random(A); assert eU[j] ne 0;
                ss -:= eU[j]*UT[j];
            end for;
            // group together the remaining coefficients
            // (to be computed as the solution to a linear system):
            L := [];
            for j in [1..n2] do
                if j in J then
                    L cat:= [j];
                end if;
            end for;
            HH := VerticalJoin([UT[L[i]] : i in [1..rU]]);
        until Determinant(HH) ne 0; // make sure the linear system admits a (unique) solution
        // compute the remaining coefficients:
        ee := ss*HH^-1;
        c := 0;
        for j in [1..n2] do
            if j in J then
                c +:= 1; eU[j] := ee[c];
            end if;
        end for;
        assert c eq rU;
        assert eU*UT eq sU;
        // perform rejection sampling:
        weU := Hweight(eU);
        //"*** wt(e_U) =", weU, ":", 1.0*(weU - (kU + (2/3)*rU)), ":", weU - (n2 - Ceiling(rU/3));
        acc := Random(0, 2^128-1) le rsU[weU + 1]*2^128; // accept with probability rsU[wt(eU)]
    until acc or aatt gt 100;
    //"***", aatt, ":", weU - (n2 - Ceiling(rU/3));
    assert aatt le 100;
    return eU;
end function;

/**
 * Compute the matrix R such that R*M is in echelon form, and also the rank of M.
 */
function Echelon(M)
    r := Nrows(M); m := Ncols(M);
    A := HorizontalJoin(M, IdentityMatrix(K, r)); // [ M | I ]
    n := Ncols(A);
    // echelonize the i-th row:
    p := 0; // pivot column
    for i in [1..r] do
```

```
        repeat
            p +:= 1;
            // force unit pivot on the p-th column:
            for k in [i..r] do
                if A[k, p] ne 0 then
                    scale := A[k, p]; // A[k, p]^-1; // no inversion if char 3
                    // swap k-th and i-th rows and
                    // normalize i-th row from the p-th column onward:
                    swap := Submatrix(A, k, p, 1, n - p + 1);
                    InsertBlock(~A, Submatrix(A, i, p, 1, n - p + 1), k, p);
                    InsertBlock(~A, scale*swap, i, p);
                    break; // done pivoting
                end if;
            end for;
        until A[i, p] eq 1 or p eq m;
        // clear the p-th column below the i-th line:
        for k in [i+1..r] do
            if A[k, p] ne 0 then
                scale := -A[k, p];
                InsertBlock(~A, Submatrix(A, k, p, 1, n - p + 1)
                    + scale*Submatrix(A, i, p, 1, n - p + 1), k, p);
            end if;
        end for;
    end for;
    // compute rank(M) from its echelon form:
    rank := 0;
    for i in [1..r] do
        if &and[A[i, j] eq 0 : j in [1..m]] then
            break;
        end if;
        rank +:= 1;
    end for;
    //assert Rank(M) eq rank;
    return Submatrix(A, 1, m+1, r, r), rank;
end function;

/**
 * Create a random (U, U+V) {\Wave} key pair.
 */
function KeyGen(rU, rV, n)
    n2 := n div 2; assert rU lt n2; assert rV lt n2;
    r := rU + rV;
    // create a matrix Hsk of form [  U  O ]:
    //                             [ -V  V ]
    U := Matrix(K, rU, n2, [Random(K) : i in [1..rU], j in [1..n2]]);
    UT := Transpose(U);
    V := Matrix(K, rV, n2, [Random(K) : i in [1..rV], j in [1..n2]]);
    VT := Transpose(V);
    O := ZeroMatrix(K, rU, n2);
    Hsk := VerticalJoin(HorizontalJoin(U, O), HorizontalJoin(-V, V));
```

```
    // create an invertible scrambling matrix S:
    repeat
        S := Matrix(K, r, r, [Random(K) : i, j in [1..r]]);
    until Determinant(S) ne 0;
    SinvT := Transpose(S^-1);
    // create a permutation matrix P:
    P := ZeroMatrix(K, n, n);
    J := {1..n};
    for i in [1..n] do
        j := Random(J); J diff:= {j};
        P[i, j] := 1;
    end for;
    // compute the public key:
    Hpk := S*Hsk*P;
    return UT, VT, SinvT, Hsk, P, Hpk;
end function;

/**
 * Sign a given syndrome s given a {\Wave} private key (S^-T, U^T, V^T, P)
 * to an error pattern of weight w +- wThreshold.
 *
 * NB: The weight threshold is used only to speed up the generation of valid signatures.
 * It is irrelevant for the attack, and could be set to zero.
 */
function Sign(SinvT, UT, VT, P, s, w, wThreshold, alpha, sigmaV, rsV, sigmaU, rsU)
    r := Ncols(s); assert Nrows(SinvT) eq r; assert Ncols(SinvT) eq r;
    n := Nrows(P); k := n - r; assert Ncols(P) eq n; assert n mod 2 eq 0; assert r lt n;
    n2 := n div 2;
    rU := Ncols(UT); kU := n2 - rU; assert Nrows(UT) eq n2; assert rU lt n2;
    rV := Ncols(VT); kV := n2 - rV; assert Nrows(VT) eq n2; assert rV lt n2;
    assert r eq rU + rV;
    // compute the privately decodable syndrome:
    ss := Eltseq(s*SinvT);
    sU := Vector(K, rU, ss[1..rU]);
    sV := Vector(K, rV, ss[rU+1..r]);
    sattv := 0;
    repeat
        sattv +:= 1;
        // find a suitable solution of eV*V^T = sV:
        eV := D_V(sV, VT, alpha, sigmaV, rsV);
        // find a dense solution of eU*U^T = sU s.t. |wt(eU, eU + eV) - w| <= wThreshold:
        sattu := 0;
        repeat
            sattu +:= 1;
            eU := D_U(sU, UT, eV, sigmaU, rsU); // only eV is different in Tsunami
            e := Vector(K, n, Eltseq(eU) cat Eltseq(eU + eV));
            wt := Hweight(e);
            //"*** wt(e) =", wt, "vs", w, ":", wt - w;
            assert wt ge n - 2*rU and wt le n;
        until Abs(wt - w) le wThreshold or sattu ge 10;
```

```
    until Abs(wt - w) le wThreshold or sattv ge 10;
    // random sigs have weight very close to, but often not exactly, w.
    assert Abs(wt - w) le wThreshold;
    // permute the error pattern:
    return e*P;
end function;

/**
 * Verify a purported signature e for syndrome s
 * under public key Hpk within a given weight threshold of w.
 *
 * NB: The weight threshold is used only to speed up the generation of valid signatures.
 * It is irrelevant for the attack, and could be set to zero.
 */
function Verify(Hpk, e, s, w, wThreshold)
    return Abs(Hweight(e) - w) le wThreshold and e*Transpose(Hpk) eq s;
end function;

/**
 * Basic {\Wave} key generation, signing & verification tests.
 */
procedure TestKeySigVer(rU, rV, n, w, wThreshold, keys, sigs, alpha, sigmaV, rsV, sigmaU, rsU)
    if keys gt 0 and sigs gt 0 then
        "**** Testing signing/verification:", keys, "key(s), ", sigs, "sig(s) per key...";
        r := rU + rV; assert n gt rU + rV;
        fail := 0;
        for key in [1..keys] do
            UT, VT, SinvT, Hsk, P, Hpk := KeyGen(rU, rV, n);
            for sig in [1..sigs] do
                s := Vector(K, r, [Random(K) : j in [1..r]]);
                e := Sign(SinvT, UT, VT, P, s, w, wThreshold, alpha, sigmaV, rsV, sigmaU, rsU);
                fail +:= (Verify(Hpk, e, s, w, wThreshold)) select 0 else 1;
            end for;
        end for;
        if fail eq 0 then
            "**** Testing complete. No failures detected.";
        else
            "**** Testing complete. Failures:", fail;
        end if;
    end if;
end procedure;

/*
 * Support infrastructure for the attack on {\Wave}.
 */

/**
 * Collect a number of legitimate signatures created under a given {\Wave} private key.
 * @return consolidated column-pairing counts inferred from those signatures.
 */
```

```
function CollectSigs(SinvT, UT, VT, P, w, wThreshold, numSig, alpha, sigmaV, rsV, sigmaU, rsU)
    r := Nrows(SinvT);
    assert Ncols(SinvT) eq r; assert Nrows(UT) eq Nrows(VT); assert Ncols(UT) + Ncols(VT) eq r;
    n := Nrows(P); assert Ncols(P) eq n;
    // collect signatures:
    sigTab := ZeroMatrix(ZZ, numSig, n);
    for sig in [1..numSig] do
        s := Vector(K, r, [Random(K) : j in [1..r]]);
        e := Sign(SinvT, UT, VT, P, s, w, wThreshold, alpha, sigmaV, rsV, sigmaU, rsU);
        sigTab[sig] := Vector(ZZ, n, [(e[j] eq 2) select ZZ!(-1) else ZZ!e[j] : j in [1..n]]);
    end for;
    // pair up matching columns:
    countTab := -Transpose(sigTab)*sigTab;
    return countTab;
end function;

/**
 * Pair up matching columns from both sides of the private (U, U+V) parity-check matrix
 * and recover an unscrambling matrix R that reveals an equivalent permuted private key,
 * given the corresponding public key and statistics inferred from the collected signatures.
 */
function RecoverR(Hpk, countTab, criticalCols)
    r := Nrows(Hpk); n := Ncols(Hpk); n2 := n div 2; assert n mod 2 eq 0;
    Z := ZeroMatrix(K, r, criticalCols);
    // compute the differences from the runner-up count to the maximum for each column:
    diffs := [[j, 0, 0] : j in [1..n]];
    for h in [1..n] do
        c := Eltseq(countTab[h]);
        maxc, j := Max(c); // find max count (i.e. the suggested pair for column h)
        diffs[h][2] := j; // position of max count
        c[j] := -n; // sentinel;
        maxt, t := Max(c); // find runner-up (i.e. the worst potential noise)
        diffs[h][3] := maxc - maxt;
    end for;
    //"***", diffs;
    Sort(~diffs, func<x, y | y[3] - x[3]>);
    done := {};
    cols := 0;
    for i in [1..n] do
        j := diffs[i][1]; // scan columns from highest to lowest S/N ratio
        if j in done then
            continue; // already paired-up (with higher S/N ratio)
        end if;
        // guess the pair for the target column:
        h := diffs[i][2];
        if h in done then
            // report failure:
            return false, Z;
        end if;
        done join:= {j, h};
```

```
        cols +:= 1;
        InsertBlock(~Z, Submatrix(Hpk, 1, j, r, 1) + Submatrix(Hpk, 1, h, r, 1), 1, cols);
        if cols eq criticalCols then
            break;
        end if;
    end for;
    assert cols eq criticalCols;
    R, rZ := Echelon(Z);
    return true, R;
end function;

/**
 * Recover the column permutation that reveals the equivalent admissible (U, U+V) code structure.
 */
function Unpermute(RHpk, rU)
    r := Nrows(RHpk); n := Ncols(RHpk); rV := r - rU; n2 := n div 2;
    assert rU lt r; assert n mod 2 eq 0;
    pair := [0 : j in [1..n]];
    for j in [1..n] do
        if pair[j] ne 0 then
            assert pair[pair[j]] eq j;
            continue; // done
        end if;
        mcolj := -Submatrix(RHpk, rU + 1, j, rV, 1);
        for p in [j+1..n] do
            if pair[p] ne 0 then
                assert pair[pair[p]] eq p;
                continue;
            end if;
            if Submatrix(RHpk, rU + 1, p, rV, 1) eq mcolj then
                assert pair[p] eq 0;
                // pair up columns j and p:
                pair[j] := p; pair[p] := j;
                break;
            end if;
        end for;
        if pair[j] eq 0 then
            // report failure:
            return false, ZeroMatrix(K, n, n);
        end if;
        assert pair[pair[j]] eq j;
    end for;
    col := 0;
    perm := [0 : j in [1..n]];
    J := {};
    for j in [1..n2] do
        repeat
            col +:= 1;
        until not col in J;
        perm[j] := col; perm[j + n2] := pair[col];
```

```
        J join:= {col, pair[col]};
    end for;
    Q := Matrix(K, n, n, [(i eq perm[j]) select 1 else 0 : i, j in [1..n]]);
    return true, Q;
end function;

/**
 * Recover a full equivalent {\Wave} private key for a given public key
 * given a suitable collection of legitimate signatures.
 */
function RecoverStructure(Hpk, rU, countTab, criticalCols)
    r := Nrows(Hpk); n := Ncols(Hpk); n2 := n div 2; assert n mod 2 eq 0;
    rV := r - rU; assert rU lt r;

    // exploit statistics to pair up columns:
    ok, R := RecoverR(Hpk, countTab, criticalCols);
    if not ok then
        // report failure:
        return false, R, ZeroMatrix(K, n, n),
            ZeroMatrix(K, n2, rU), ZeroMatrix(K, n2, rU), ZeroMatrix(K, n2, rV);
    end if;

    // unscramble the public key and unpermute it into an equivalent private key:
    Hbk := R*Hpk;
    ok, Q := Unpermute(Hbk, rU);
    if not ok then
        // report failure:
        return false, R, Q,
            ZeroMatrix(K, n2, rU), ZeroMatrix(K, n2, rU), ZeroMatrix(K, n2, rV);
    end if;
    Hbk *:= Q;
    ok := IsZero(Submatrix(Hbk, rU+1, 1, rV, n2) + Submatrix(Hbk, rU+1, n2+1, rV, n2));
    if not ok then
        // report failure:
        return false, R, Q,
            ZeroMatrix(K, n2, rU), ZeroMatrix(K, n2, rU), ZeroMatrix(K, n2, rV);
    end if;
    BT := Transpose(Submatrix(Hbk, 1, n2+1, rU, n2)); // B^T
    DT := Transpose(Submatrix(Hbk, 1, 1, rU, n2)) + BT; // (A + B)^T
    //rD := Rank(DT); assert rD eq rU;
    WT := Transpose(Submatrix(Hbk, rU+1, n2+1, rV, n2)); // W^T
    //rW := Rank(WT); assert rW eq rV;
    return true, R, Q, BT, DT, WT;
end function;

/**
 * Forge a {\Wave} signature for an arbitrarily given syndrome,
 * given an equivalent trapdoor corresponding to the legitimate public key.
 */
function ForgeSignature(R, BT, DT, WT, Q, Hpk, s, w, alpha, sigmaV, rsV, sigmaU, rsU)
```

```
    r := Nrows(Hpk); n := Ncols(Hpk); n2 := n div 2; rU := Ncols(BT);
    assert Nrows(Q) eq n; assert Ncols(Q) eq n; assert n mod 2 eq 0;
    rV := r - rU; kV := n2 - rV; assert rU lt r;
    sp := Eltseq(s*Transpose(R));
    sU := Vector(K, rU, sp[1..rU]);
    sV := Vector(K, rV, sp[rU+1..r]);
    // find a suitable solution of eV*V^T = sV:
    attv := 0;
    repeat
        attv +:= 1;
        eV := D_V(sV, WT, alpha, sigmaV, rsV);
        // find a dense solution of e_U D^T = s_U - e_V B^T.
        ss := sU - eV*BT;
        attu := 0;
        repeat
            attu +:= 1;
            eU := D_U(ss, DT, eV, sigmaU, rsU); // only eV is different in Tsunami
            e := Vector(K, n, Eltseq(eU) cat Eltseq(eU + eV));
            wt := Hweight(e);
            assert wt ge n - 2*rU and wt le n;
        until wt eq w or attu ge 32;
    until wt eq w or attv ge 32;
    assert wt eq w;
    // permute the error pattern:
    e := e*Transpose(Q);
    return e;
end function;

// some toy parameters, for testing purposes only:
//kU := 17; kV := 12; k := kU + kV; n := 38; r := n - k;
//kU := 23; kV := 16; k := kU + kV; n := 52; r := n - k;
//kU := 46; kV := 32; k := kU + kV; n := 104; r := n - k;
//kU := 92; kV := 64; k := kU + kV; n := 208; r := n - k;
//kU := 230; kV := 161; k := kU + kV; n := 518; r := n - k; // avg numSig: 3400
//kU := 460; kV := 322; k := kU + kV; n := 1034; r := n - k;
//kU := 920; kV := 644; k := kU + kV; n := 2068; r := n - k;
//kU := 1150; kV := 805; k := kU + kV; n := 2586; r := n - k;
//kU := 1840; kV := 1288; k := kU + kV; n := 4132; r := n - k;

// actually proposed 128-bit level parameters:
kU := 2299; kV := 1609; k := kU + kV; n := 5172; r := n - k;

n2 := n div 2;
rU := n2 - kU;
rV := n2 - kV;
w := n - Ceiling(2*rU/3);
wThreshold := Ceiling((n - w)/20); // 5% tolerance, just for practicality of legitimate signatures
"n  =", n,  ": w  =", w;
"kU =", kU, ": rU =", rU;
"kV =", kV, ": rV =", rV;
```

```
"k  =", k,  ": r  =", r;
"threshold:", wThreshold;
criticalCols := Ceiling(rU + Log(3, rU)); // minimum = rU,  maximum = n2
"*** criticalCols:", criticalCols;

Tsunami := false; // set this to true to test the Tsunami variant

if Tsunami then
    "Testing the Tsunami variant...";
    zeta := (1/2)*(1 + (1/3)*rV/kV); // Tsunami co-density
    alpha := zeta;
    sigmaV := 2^-64*n; // should not exceed 2^{-lambda/2}*n at the lambda-bit security level
    sigmaU := sigmaV; // idem
else
    "Testing the original Wave...";
    alpha := 0.545;
    sigmaV := (kV eq 1609) select 18.81 else Sqrt(kV)/2; // approximate sigmaV for toy parameters
    sigmaU := sigmaV; // NB: the Wave paper is unclear here; this is just a guess! <<<<<<<<<<<<<<<<<<
end if;

"computing acceptance rate vector rsV...";
time rsV := MakeRsV(n, w, kV, alpha, sigmaV);
"rsV computed";

"computing acceptance rate vector rsU...";
rsU := [1 : i in [0..n2]]; // PROVISIONALLY!
"rsU computed";

TestKeySigVer(rU, rV, n, w, wThreshold, 10, 10, alpha, sigmaV, rsV, sigmaU, rsU);
time TestKeySigVer(rU, rV, n, w, wThreshold, 1, 100, alpha, sigmaV, rsV, sigmaU, rsU);

numSig0 := 5500;
topSigs := 15000;
// in the unlikely case of attack failure, slightly increase topSigs
// (NB: Tsunami resists this attack, increasing topSigs is ineffective against it)
succ := 0;
fail := 0;
loop := 100;
deltaSig := 100;
avgSigs := 0;
sdvSigs := 0;
minSigs := topSigs;
maxSigs := 0;
time for test in [1..loop] do
    "================ test #", test, "-- success ratio so far:",
    (test - 1 - fail), "out of", test - 1,
    "-- avg #signatures:", (test gt 1) select Round(avgSigs/(test-1)) else 0,
    "+-", (test gt 1) select Round(Sqrt(sdvSigs/(test-1) - (avgSigs/(test-1))^2)) else 0,
    (test gt 1) select "["*Sprint(minSigs)*".."*Sprint(maxSigs)*"]" else "[0 .. 0]";
```

```
    // create a sample key pair:
    UT, VT, SinvT, Hsk, P, Hpk := KeyGen(rU, rV, n);

    // break Wave:
    "collecting", numSig0, "signatures..."; // in chunks of deltaSig signatures
    countTab := CollectSigs(SinvT, UT, VT, P, w, wThreshold, deltaSig, alpha,
        sigmaV, rsV, sigmaU, rsU);
    numSig := deltaSig;
    while numSig lt numSig0 do
        moreSigs := Min(deltaSig, numSig0 - numSig);
        countTab +:= CollectSigs(SinvT, UT, VT, P, w, wThreshold, moreSigs, alpha,
            sigmaV, rsV, sigmaU, rsU);
        numSig +:= moreSigs;
    end while;
    repeat
        "recovering structure...";
        ok, R, Q, BT, DT, WT := RecoverStructure(Hpk, rU, countTab, criticalCols);
        if not ok then
            if numSig ge topSigs then
                break; // failure
            end if;
            "collecting", deltaSig, "more signatures...";
            countTab +:= CollectSigs(SinvT, UT, VT, P, w, wThreshold, deltaSig, alpha,
                sigmaV, rsV, sigmaU, rsU);
            numSig +:= deltaSig;
        end if;
    until ok;
    avgSigs +:= numSig;
    minSigs := Min(minSigs, numSig);
    maxSigs := Max(maxSigs, numSig);
    sdvSigs +:= numSig^2;
    if not ok then
        "recover failure!";
        fail +:= 1;
        continue;
    end if;

    "forging signature...";
    s := Vector(K, r, [Random(K) : j in [1..r]]); // target syndrome
    e := ForgeSignature(R, BT, DT, WT, Q, Hpk, s, w, alpha, sigmaV, rsV, sigmaU, rsU);
    ok := Verify(Hpk, e, s, w, 0); // NB: zero tolerance for the signature weight here
    if not ok then
        "forging failure!";
        fail +:= 1;
        continue;
    end if;
    "success!";
end for;
"*** success ratio:", (loop - fail), "out of", loop, "("*Sprint((loop - fail)/loop)*")",
    "-> failures:", fail;
```

```
avgSigs /:= loop;
sdvSigs := Sqrt(sdvSigs/loop - avgSigs^2);
"*** avg collected signatures:", Round(avgSigs), "+-", Round(sdvSigs),
    "["*Sprint(minSigs)*".."*Sprint(maxSigs)*"]";
```