Wave Signature Demonstration Software, v1

Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich

October 28, 2019

Preliminary comment. The software is by no mean a production software. The implementation is meant to be a reference for the algorithms but is not protected in any way against any sort of side-channel attack. It is not meant to be deployed, even in part, in a real life security system. Its purpose is to demonstrate the feasibility of the Wave signature scheme primitives: key generation, signature, verification. It may also be used to evaluate the security features of the scheme, in particular the leakage resilience.

1 Using the Software

The software was developed in standard C for Linux with gcc (version 5.4.0). We did not try it on other platforms.

1.1 Building

A `Makefile` is provided. Typing `make` will generate 4 executables `keygen`, `sign`, `verif`, and `showpr`.

1.2 Package Description

The package contains all source files, a Makefile, a directory `./Data/` which contain one subdirectory for each parameter set. In this version there are 4 parameter sets. For each parameter set, pre-

identifier		128g	96g	80g	64g
block length	n	8492	6368	5308	4246
error weight	w	7980	5984	4988	3990
dimension of U	k_U	3558	2668	2224	1779
dimension of V	k_V	2047	1535	1280	1024
gap[1]	d	81	61	50	40
security (bits)	λ	128	96	80	64

Table 1: Parameter Sets (version 1)

computed data is stored in the appropriate directory, *e.g.* `./Data/128g/wave_precomp128g.dat` for the parameter identifier "`128g`". All files, keys, signatures, log, ... relevant to a particuliar identifier are saved in this directory.

---

[1]The gap in the quantity d introduced in [1, §4,Proposition 6], its value is $\approx \lambda/\log_2 3$ for λ bits of security

External Files. The file `keccak.c` contains an implementation of SHA3 and comes from the KECCAK team website[2]. The files `cmdline.[ch]` are generated from `wave.ggo` with gengetopt[3].

1.3 `keygen`

The command `keygen -i 128g -k 1234` will produce a key pair corresponding to the parameters identifier "128g" with a random generator seeded with the integer 1234. Without explicit `'-k'` the seed is 0.

The public and secret keys are saved respectively in the files `./Data/128g/wave_pk128g_1234.dat` and `./Data/128g/wave_sk128g_1234.dat`.

1.4 `sign`

The command `sign -i 128g -k 1234 -m 99 -n 1000` will generate 1000 signatures of messages seeded with all integers in $[99, 1098]$ using the key seeded by 1234 and with the parameters corresponding to the identifier "128g". The keys must have been generated previously by `keygen`.

Signatures are saved in `./Data/128g/sign_128g_1234_99-1098.dat`.

1.5 `verif`

The command `verif -i 128g -k 1234 -f ./Data/128g/sign_128g_1234_99-1098.dat` will verify the signature contained in the file given by `'-f'` with the key of identifier given by `'-i'`. The public key must have been generated previously by `keygen`.

1.6 `showpr`

Print all rejection sampling data (see next section).

1.7 Random Number Generation

All the 'randomness' is generated in a deterministic way from Keccak (SHA3). This holds for keys and messages which should thus be the same (for given seeds) regardless of the platform. This also holds for the internal randomness of the signature primitive. This was done to allow reproducibility of signature generation. Consequently, the software is not fully compliant with the specification. This should not make a difference for a demo software.

2 Rejection Sampling Data

All the data used for rejection sampling is precomputed as described in the long version of the paper [1].

---

[2]`https://keccak.team/`
[3]`https://www.gnu.org/software/gengetopt`

2.1 Internal Distributions

Internal discrete distributions appear in each of the two decoders, $\mathcal{D}_V$ in DECODEV and $\mathcal{D}_U^t$ in DECODEU. The latter is parameterized by t, the Hamming weight of DECODEV output.

Those distribution are very close to generalized Laplace distribution as defined in [1, §4.3]. Let us denote $\tilde{\mathcal{D}}_V$ and $\tilde{\mathcal{D}}_U^t$ those ditributions, and X denote a random variable following one of them.

We keep only the values ℓ such that the probability $\mathbb{P}(X = \ell)$ exceeds $2^{-\lambda}$ where λ is the target security level in bits (*e.g.* 128). We sort the remaining value in (almost) increasing order $\ell_0, \ldots, \ell_{N-1}$ and we compute the cumulated probabilities

$$\tilde{S}_0 = \tilde{P}_0, \tilde{S}_i = \tilde{S}_{i-1} + \tilde{P}_i, \text{ where } \tilde{P}_i = \mathbb{P}(X = \ell_i)$$

We round the cumulated probabilities $\tilde{S} \in \{\tilde{S}_0, \ldots \tilde{S}_{N-2}, \tilde{S}_{N-1} = 1\}$ with (up) to 24 bits of precision

$$\tilde{S} \approx S = M \, 2^{-W}, \text{ with } \begin{cases} 2^{23} \leq M < 2^{24}, W = 23 + \lceil -\log_2(\tilde{S}) \rceil & \text{if } \tilde{S} \geq 2^{-\lambda+24}, \\ 0 < M < 2^{24}, W = \lambda & \text{else.} \end{cases}$$

The number of significant bits of S is $\max(24, \lambda - \log_2(2\tilde{S}))$. We represent any such number with 5 unsigned bytes $(\omega, B_0, B_1, B_2, B_3)$

$$S = M \, 2^{-W} = 256^{-\omega} \left(B_0 \, 256^{-1} + B_1 \, 256^{-2} + B_2 \, 256^{-3} + B_3 \, 256^{-4} \right), \omega = \lceil (W+1)/8 \rceil - 4$$

Finally each (cumulated) distribution is stored as an array of size N containing the (ordered) (ℓ_i, S_i) where each S_i is represented by 5 bytes as described above. We define the probabilities

$$P_0 = S_0, P_i = S_i - S_{i-1}$$

and a new random variable Y such that $\mathbb{P}(Y = \ell_i) = P_i$. The variable Y will follow new distributions, $\mathcal{D}_V$ or $\mathcal{D}_U^t$, according to the one used for X, to be used in the decoders. They are very close but not equal to the original distributions based on Laplace.

Note that having $\tilde{P}_i$ in increasing order (or close to that) was a important precaution to guaranty that P_i is close to $\tilde{P}_i$.

2.2 Rejection Vectors

Rejection vectors will be computed as defined in the paper [1, §4]. The distributions used to compute the coefficients must be the distributions deriving from P_i rather than $\tilde{P}_i$, that this the distribution we obtained after rounding the probabilities with a finite precision (here 24 bits). Failing to do that will introduce a small bias in the signatures and may allow a leakage attack.

Also, the rejection vector must be computed with enough precision to guaranty the absence of bias. We computed the coefficients with λ bits of precision where λ is the target security level in bits (*e.g.* 128). We used the multiprecision floating point arithmetic of GNU MP.

References

[1] Debris-Alazard, T., Sendrier, N., Tillich, J.P.: Wave: A new family of trapdoor one-way preimage sampleable functions based on codes. Cryptology ePrint Archive, Report 2018/996 (October 2019), https://eprint.iacr.org/2018/996