From: Ward Beullens <Ward.Beullens@esat.kuleuven.be>
Sent: Monday, January 15, 2018 4:55 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: WalnutDSA
Attachments: walnutdsa.pdf

Dear all,

The paper [1] describes an attack on an earlier version of WalnutDSA. In response, WalnutDSA was adapted to block this attack by using two braids as private key instead of one. The attached pdf describes a way to circumvent this adaptation, and shows how to use the method of [1] to attack the adapted scheme.

I have communicated with the designers, who have confirmed that the attack works. However, the new attack has the same limitation as the attack of [1] that the forged signatures are much longer than signatures made with the legitimate signing algorithm. WalnutDSA implicitly imposes a bound on the length of the signatures by requiring that the length of a signature (in terms of the number of artin generators) is encoded in a two byte value. The forgeries of [1] are much longer than 2^16, so the WalnutDSA scheme is not (yet) broken.

The attack paper [1] mentions "If an efficient algorithm to compute short factorizations exists, the increase in parameters q and N needed to achieve a sufficient level of security would then make WalnutDSA unsuitable for embedded devices." This statement now also holds for the version of WalnutDSA that is submitted to NIST. Moreover, the security of Walnut also depend on the hardness of this problem:

Given a long braid s and a pair (M,sigma), find a short (e.g. shorter than 2^16) braids s' such that (M,sigma) * s = (M,sigma) * s'.

On a separate note: I believe the security proof to be flawed. (I did not discuss this with the designers)

The reduction uses a EUF-CMA adversary to solve their hard problem (the REM problem). However, the EUF-CMA adversary can make signing queries, and the security proof does not describe a method to respond to these queries. Therefore, it seems like the proof only works for a key-only attack, where no signing queries have to be answered.

Moreover, the proof does not work in the quantum random oracle model, makes some assumptions and imposes restrictions on the attacker, so even for key-only attacks the practical value of the security proof seems very limited.

Kind regards,
Ward

[1] Hart, D., Kim, D., Micheli, G., Perez, G. P., Petit, C., & Quek, Y.
A Practical Cryptanalysis of WalnutDSATM.

WalnutDSA

Ward Beullens

January 15, 2018

1 Definitions

We use the notation of the WalnutDSA white paper[1]. We fix a set of T values throughout the document, and we denote by $*$ the action of B_N on $GL(N, \mathbb{F}_q) \times S_n$ of E-multiplication with respect to T.

Definition 1. *The* **hash** *of a message $m \in \{0,1\}^*$ is $E(H(m)) \in B_N$, which is a pure braid by construction.*

Definition 2. *For a braid $s \in B_N$ we define $P(s)$ to be equal to*

$$(\mathbb{1}_N, e) * s \in GL(N, \mathbb{F}_q) \times S_N.$$

Definition 3. *A signature $s \in B_N$ is valid for a message with hash g for the public key $(P(w), P(w'))$ if and only if*

$$P(w) * s = P(g) * w',$$

where $*$ stands for E-multiplication with respect to some T-values.

Remark. *The above definition of what it means for a signature to be valid is not completely identical to the definition of the WalnutDSA white paper, where only the matrix part of the above equation is required to hold. However, for all signatures that that are produced by the legitimate Walnut signing algorithm, the permutation part of the above equation also holds. Therefore we can assume this stricter definition.*

2 Two properties of WalnutDSA

The Walnut digital signature algorithm exhibits two interesting properties. Together, these properties generalize theorem 4 of [2]:

Theorem 1. *Suppose that $w, w', s, g \in B_N$ are braids such that s is a valid signature for a messages with hash g for the public key $(P(w), P(w'))$. Then we have that s^{-1} is a valid signature for any message with hash g^{-1} for the public key $(P(w'), P(w))$.*

Proof. By definition we have

$$P(w) * s = P(g) * w'.$$

Acting on this by s^{-1} from the left and using the definition of P we get

$$\begin{aligned}(\mathbb{1}_N, e) * w &= (\mathbb{1}_N, e) * g * w' * s^{-1} \\ &= (CB(g)_{\downarrow T}, e) * w' * s^{-1}\,,\end{aligned}$$

where the second equality uses the definition of E-multiplication and the fact that g is a pure braid. Multiplying the matrix part of this equality by $(CB(g)_{\downarrow T})^{-1}$ from the left (which is compatible with $*$) we get

$$(CB(g^{-1})_{\downarrow T}, e) * w = (\mathbb{1}_N, e) * w' * s^{-1}\,,$$

or equivalently

$$P(g^{-1}) * w = P(w') * s^{-1}$$

which shows that s^{-1} is a valid signature for any message with hash g^{-1} for the public key $(P(w'), P(w))$. □

Theorem 2. *Suppose that $w, w', w'', s_1, s_2, g_1, g_2 \in B_N$ are braids such that s_1 is a signature for a message with hash g_1 that is valid for the public key $(P(w), P(w'))$, and such that s_2 is a signature for a message with hash g_2 that is valid for the public key $(P(w'), P(w''))$. Then we have that $s_1 s_2$ is a valid signature for any message with hash $g_1 g_2$ for the public key $(P(w), P(w''))$.*

Proof. using the fact that s_1 is a valid signature for g_1, and using the definition of E-multiplication we get

$$\begin{aligned}P(w) * s_1 * s_2 &= P(g_1) * w' * s_2 \\ &= (CB(g_1)_{\downarrow T} \cdot CB(w')_{\downarrow T}{}^{\cdot^{\sigma_{w'}}} (CB(s_2))_{\downarrow T}, \sigma_{w'} \circ \sigma_{s_2})\,,\end{aligned}$$

where $\sigma_{w'}$ denotes the permutation of w'. Using that s_2 is a signature for g_2 for the public key $(P(w'), P(w''))$ we can continue

$$\begin{aligned}P(w) * s_1 * s_2 &= (CB(g_1)_{\downarrow T} \cdot CB(g_2)_{\downarrow T} \cdot CB(w'')_{\downarrow T}, \sigma_{w''}) \\ &= P(g_1 g_2) * w''\,,\end{aligned}$$

which shows that $s_1 s_2$ is a valid signature for any message with hash $g_1 g_2$ for the public key $(P(w), P(w''))$. □

3 Attack on WalnutDSA

Our attack uses the factorization attack of Hart et al. [2] on an earlier version of WalnutDSA as a black box. The earlier version of WalnutDSA uses only one braid as a private key. Hence, their attack works in the case $w = w'$.

Assumption. *Given a long enough list of signatures $s_1, \cdots, s_k$ for messages with hashes $g_1, \cdots, g_k$ respectively that are valid for a public key $(P(w), P(w))$, and a target hash g, the attack of [2] can forge a signature s that is valid for any message with hash g for the same public key.*

3.1 Description of the attack

We will show how an attacker can use a number of valid signature-hash pairs $(s_1, g_1), \cdots, (s_k, g_k)$ that are valid for a public key $(P(w), P(w'))$ to forge a signature for any message that is valid for the same public key.

For any $i, j \in \{1, \cdots, k\}$ we have that $s_i s_j^{-1}$ is a valid signature for any message with hash $g_i g_j^{-1}$ for the public key $(P(w), P(w))$. This follows easily from theorems 1 and 2. Therefore, by collecting k signed messages for $(P(w), P(w'))$, the attacker can obtain $k(k-1)/2$ message-hash pairs for $(P(w), P(w))$. This fact, in combination with the attack of [2] allows the attacker to forge a signature for any hash value g that is valid under the public key $(P(w), P(w))$.

Let g be the hash of a message that the attacker wants to forge a signature for. As per the previous paragraph, the attacker can obtain a signature s for the hash $g g_1^{-1}$, which is valid for the public key $(P(w), P(w))$. Theorem 2 then says that $s s_1$ is a valid signature for any message with hash $g g_1^{-1} g_1 = g$ for the public key $(P(w), P(w'))$, so this is a forgery that the attacker was looking for.

3.2 Limitations of the attack

The new attack has the same limitations as the attack of [2], and the countermeasures proposed in [2] should suffice to also block the new attack. The new attack produces forged signatures that are roughly twice as long as the forgeries of [2], but it is easier to collect many signature-hash pairs, which can help to find shorter forgeries.

References

[1] Iris Anshel, Derek Atkins, Dorian Goldfeld, and Paul E Gunnells. Walnutdsa (tm): A quantum resistant group theoretic digital signature algorithm. *IACR Cryptology ePrint Archive*, 2017:58, 2017.

[2] Daniel Hart, DoHoon Kim, Giacomo Micheli, Guillermo Pascual Perez, Christophe Petit, and Yuxuan Quek. A practical cryptanalysis of walnutdsatm.

From: Derek Atkins <datkins@securerf.com>
Sent: Tuesday, January 16, 2018 4:56 PM
To: ward.beullens@student.kuleuven.be; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: WalnutDSA

Dear Ward, All,

As pointed out by Ward Beullens,it is possible with his method to generate extremely long forged signatures estimated at 2^30 Artin generators long (which is ~2^32 bits). The WalnutDSA specification as submitted (which Ward Buellens notes), limits signature lengths. We have already tested the Dehornoy method, as suggested by both Ward Beullens and [1], to shorten forged signatures of this form and the result was only a 10% reduction, not nearly enough to reach valid signature lengths (even a 90% reduction wouldn't come close). Therefore, WalnutDSA, as specified in the NIST submission, remains a viable method.

In regards to the security proof, in our paper we define a forger to be a randomized algorithm that can make hash queries to a random oracle and signature queries to a simulator that does not know Priv(S) but can simulate an honest signer. In our paper, we defined a signature query to be the message and the public key of the signer. The response to the query is a valid signature.

The forger is trying to generate a signature for a message that hasn't been queried before while the simulator is trying to break the hard problem of reversing E-multiplication. It is assumed that the simulator can simulate proper responses to the forger's signature queries in a way that is indistinguishable from query responses by a true signer (who knows the private key).

It is clearly stated in our paper that we assume the signatures produced by the forger are of the same type as those produced by a true signer, i.e., lie in a certain double coset of the braid group. This is a restrictive assumption which occurs because of the non-commutativity of the braid group and does not arise in security proofs for cryptosystems which make use of cyclic groups, for example. We also remark in our paper that Koblitz and Menezes [2] point out that "although it is a common approach in modern security proofs to restrict the capabilities of the adversary, it is important to show that certain classes of attacks can be ruled out."

Therefore, we submit that our security proof is not flawed in light of the perspective put forth by Koblitz and Menezes.

-- The WalnutDSA Team

[1] D. Hart; D. Kim; G. Micheli; G. Pascual Perez; C. Petit; Y. Quek, A Practical Cryptanalysis of WalnutDSA, preprint 2017.

[2] N. Koblitz; A. Menezes, Another look at "provable security," J. Cryptol. 20, 3–37 (2007).

On Mon, 2018-01-15 at 10:57 +0100, Ward Beullens wrote:

```
Dear all,

The paper [1] describes an attack on an earlier version of WalnutDSA. In
```

From: Ward Beullens <ward.beullens@student.kuleuven.be>
Sent: Thursday, January 18, 2018 11:28 AM
To: Derek Atkins; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: WalnutDSA

Dear Derek, All,

I agree that WalnutDSA is not broken yet, but I don't think the attack is insignificant either. After all, the WalnutDSA team decided it was worthwhile to modify the original scheme in an attempt to block this attack, even at the cost of doubling the public and private key sizes. (Will this modification be kept, now that is is clear that it does not really block this attack?)

About the security proof: I don't think my concern was addressed. The proof uses a EUF-CMA forger, but does not give a way to answer the signing queries of this forger. You say "we define a forger to be a randomized algorithm that can make hash queries to a random oracle and signature queries to a simulator that does not know Priv(S) but can simulate an honest signer."

This seems very peculiar to me, do you assume the existence of such a simulator? If this assumption were true, then WalnutDSA is not secure.

The security proof mentions "The Forger F is defined to be a randomized algorithm that on input a message m $\in\{0, 1\}^*$, a signers public key Pub(S), and a coin ρ, outputs a 4-tuple (m, h, g_ρ,s) , where h = H(m) and g_ρ = G_ρ(V) and V ← Cloak, s ← DC_m,V,H,G. It is assumed that the probability that (m, h, gρ,s) is a valid WalnutDSA-I signature is non-negligible."

This looks like a universal (the message can be freely chosen), key-only attack (there is no mention of signing queries). How does this relate to the claim that Walnut is EUF-CMA secure?

Kind regards,
Ward

Op 16/01/2018 om 22:55 schreef Derek Atkins:

> Dear Ward, All,
>
> As pointed out by Ward Beullens,it is possible with his method to generate extremely long forged signatures estimated at 2^30 Artin generators long (which is ~2^32 bits). The WalnutDSA specification as submitted (which Ward Buellens notes), limits signature lengths. We have already tested the Dehornoy method, as suggested by both Ward Beullens and [1], to shorten forged signatures of this form and the result was only a 10% reduction, not nearly enough to reach valid signature lengths (even a 90% reduction wouldn't come close). Therefore, WalnutDSA, as specified in the NIST submission, remains a viable method.
>
> In regards to the security proof, in our paper we define a forger to be a randomized algorithm that can make hash queries to a random oracle and signature queries to a simulator that does not know Priv(S) but can simulate an honest signer. In our paper, we defined a signature query to be the message and the public key of the signer. The response to the query is a valid signature.

From: Blackburn, S <S.Blackburn@rhul.ac.uk>
Sent: Monday, January 22, 2018 5:55 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: WalnutDSA
Attachments: Walnut_private_key_2.pdf

Dear All,

There are fewer than 2^512 possibilities for each half of the public key in WalnutDSA. This makes the scheme vulnerable to `square root' attacks (using an approach similar to Pollard-rho). It is recommended that parameter sizes should be increased so this approach is no longer feasible.

More details are given in the attached pdf, which is also available at the following link:

https://www.dropbox.com/s/dyje618qfb4zvsj/Walnut_private_key_2.pdf?dl=0

Yours,

Simon Blackburn

Simon R. Blackburn
Professor of Pure Mathematics
Department of Mathematics
Royal Holloway University of London
Egham, Surrey TW20 0EX, United Kingdom
Tel: (+44) (0)1784 443422
E-mail: S.Blackburn@rhul.ac.uk
Web: http://www.ma.rhul.ac.uk/sblackburn

Recovering a Private Key in WalnutDSA

Simon R. Blackburn
Department of Mathematics
Royal Holloway University of London
Egham, Surrey TW20 0EX, United Kingdom
e-mail: s.blackburn@rhul.ac.uk.

January 22, 2018

Abstract

There are fewer than 2^{512} possibilities for each half of the public key in WalnutDSA. This makes the scheme vulnerable to 'square root' attacks (using an approach similar to Pollard-rho). It is recommended that parameter sizes should be increased so this approach is no longer feasible.

1 Introduction

WalnutDSA(TM) is a digital signature scheme proposed by SecureRF for the NIST Post-quantum standardization call, using techniques from braid group theory. Parameters for two security levels, at 128- and 256-bits, have been defined, as well as 40-bit test parameters. This note describes an attack that recovers an equivalent private key of a signature from the corresponding public key. Analysis is given to support the claim that the attack recovers an equivalent private key for 256-bit parameters using significantly fewer than 2^{256} operations (one reasonable estimate being as little as 2^{168} operations). The 128-bit parameters are possibly also affected, with one estimate for deriving a private key being as little as 2^{108} operations.

The current version of WalnutDSA is specified by Anshel, Atkins, Goldfeld and Gunnells in [1, 3]. There has been a cryptanalysis due to Hart, Kim,

Micheli, Perez, Petit and Quek [5] of the original proposed scheme [2], which caused WalnutDSA to be significantly revised by introducing two braids (rather than the original one) as the private key. A recent note due to Beullens [4] shows that the new version of WalnutDSA remains vulnerable to the attack of Hart *et al.*

We will use the notation in [1, 3] without comment, and we will assume knowledge of these papers.

2 Recovering the private key

Let X be the set of pairs (M, σ), where M is an $N \times N$ matrix over $\mathbb{F}_q$, and where $\sigma \in S_N$ is a permutation of $\{1, 2, \ldots, N\}$. We write $1 \in X$ for the element (I, e), where I is the identity $N \times N$ matrix, and where e is the identity permutation. The braid group B_N acts on X (on the right) via E-multiplication: for $x \in X$ and $g \in B_N$, we write $x * g \in X$ for the E-multiplication of x and g. We write Ω for the orbit of 1 under this action, so $\Omega = 1 * B_N$.

The private key consists of two N-string braids $S, S' \in B_N$, each a product of Artin generators of length ℓ. The public key consists of the pair of elements $(1 * S, 1 * S') \in \Omega^2$.

The appendix describes an algorithm (using standard techniques) that takes as input an element $\omega \in \Omega$, and outputs an element $g \in B_N$ such that $1 * g = \omega$. The algorithm runs in $\sqrt{|\Omega|}$ operations, and uses insignificant memory. We will apply this algorithm twice, first with $\omega = 1*S$ and secondly with $\omega = 1 * S'$, to recover an equivalent private key $(\tilde{S}, \tilde{S'}) \in B_N \times B_N$ for WalnutDSA. Each braid in the equivalent private key is a product of length 2λ in the Artin generators for some integer λ. We choose λ to be as small as possible so that the elements $1g$ are approximately uniformly distributed in Ω when g is a length λ word in the Artin generators. An accurate value for λ seems hard to determine, but we give a heuristic estimate for λ using the following argument. A necessary condition for λ is that the number of braids of length λ must be at least $|\Omega|$. Using the approximations in Anshel *et al.* [3] for the number of braids of length λ, we set λ to be the smallest integer such that

$$|\Omega| < (2^\lambda/\lambda)(N-1)\binom{\lambda - 2 + N}{N-1}.$$

The table below gives the parameters of the attack for 40-, 128- and 256-bit security levels (SL). The size of Ω is given as a range of possible values, computed as follows. The upper bound is $N!\,q^{N(N-1)}$, as there are $N!$ choices for the permutation σ and $q^{N(N-1)}$ choices for an $N \times N$ matrix over $\mathbb{F}_q$ with last row $(0, 0, \ldots, 0, 1)$. The lower bound of $N!\,q^{N(N-3)}$ uses the estimate for the number of choices for M that is used by the authors of the scheme [3, Section 11] in their security analysis.

SL	N	q	$\lvert\Omega\rvert$	$\sqrt{\lvert\Omega\rvert}$	ℓ	2λ
40	8	2^5	$[2^{215.3}, 2^{295.3}]$	$[2^{108}, 2^{148}]$	132	$[360, 514]$
128	8	2^5	$[2^{215.3}, 2^{295.3}]$	$[2^{108}, 2^{148}]$	132	$[360, 514]$
256	8	2^8	$[2^{335.3}, 2^{463.3}]$	$[2^{168}, 2^{232}]$	287	$[592, 842]$

3 Conclusions

The attack presented here recovers an equivalent 256-bit private key using significantly fewer than 2^{256} operations (between 2^{168} and 2^{232} operations). It is quite possible that a 128-bit key might also be recovered in fewer than 2^{128} operations using this approach (as little as 2^{108} operations is one estimate). The 40-bit keys (for testing) are unaffected by this attack.

Experiments might provide more precise length estimates, but it seems likely that the equivalent private key will be comparable in length to the true private key (at most a factor of 4 longer if we use the estimates of 2λ above). This is likely to lead to forged signatures of about the right length (particularly if we reduce the length of cloaking elements, which we can do as we no longer care to keep keys secret). (Previous attacks [4, 5] produce signatures of length approximately 2^{25}. These signatures are significantly longer than signatures produced with the private key, and so would fall foul of the limit on a signature length proposed in the specification.)

Ward Beullens (who kindly looked at a draft of this note) points out that the running time of the attack can be improved by a factor of $\sqrt{N!}$, at the expense of possibly slightly longer forgeries, by modifying the algorithm in the appendix to take the application setting into account. The algorithm in the appendix should be modified to choose $g_1, g_2 \in G$ so that $1g_1 \in \Omega$ and $\omega g_2^{-1} \in \Omega$ always have a trivial permutations associated with them.

A minor additional improvement to the algorithm in the appendix can be made by mandating that the determinant of the matrices associated with $1g_1 \in \Omega$ and ωg_2^{-1} are both 1.

Recommendation: The attack presented here is exponential, and so can be avoided by increasing parameter sizes. To prevent this attack for k-bit security level, parameters should be chosen so that $q^{N(N-3)-1} > 2^{2k}$. So revising parameters at both 128-bit and 256-bit security levels is recommended.

The authors of WalnutDSA have looked a draft of this note, and confirmed that they believe the attack to be correct. They suggest increasing N from 8 to 10 to prevent this attack.

References

[1] Iris Anshel, Derek Atkins, Dorian Goldfeld and Paul E. Gunnells, 'The Walnut digital signature algorithm (TM) specification', *NIST post-quantum cryptography standardization project*. `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[2] Iris Anshel, Derek Atkins, Dorian Goldfeld and Paul E. Gunnells, 'WalnutDSA (TM): A quantum-resistant digital signature algorithm', *Cryptology ePrint archive*, Report 2017/1160, 18 September 2017. `https://eprint.iacr.org/2017/058`.

[3] Iris Anshel, Derek Atkins, Dorian Goldfeld and Paul E. Gunnells, 'WalnutDSA (TM): A quantum-resistant digital signature algorithm', *Cryptology ePrint archive*, Report 2017/1160, 30 November 2017. `https://eprint.iacr.org/2017/058`.

[4] Ward Beullens, 'WalnutDSA', *NIST Official Comment*, 15 January 2018.

[5] Daniel Hart, DoHoon Kim, Giacomo Micheli, Guillermo Pascual Perez, Christophe Petit and Yuxuan Quek, 'A practical cryptanalysis of WalnutDSA (TM)', *Cryptology ePrint archive*, Report 2017/1160, 29 November 2017. `https://eprint.iacr.org/2017/1160`.

A An algorithm from computational group theory

Let Ω be a finite set, and let G be a group that acts transitively on Ω (on the right). Let $1 \in \Omega$ be fixed. Suppose we are given $\omega \in \Omega$, and we wish to

find $g \in G$ such that $1g = \omega$. We present an algorithm (using standard cryptographic techniques akin to the Pollard-rho discrete logarithm algorithm) that solves this problem using $O(\sqrt{|\Omega|})$ operations (in expectation).

First, choose a pseudorandom number generator r that takes a seed $x \in \Omega$ and outputs a pair $r(x) = (g, a)$, where $g \in G$ and $a \in \{0, 1\}$. (In our application, we would fix a positive integer λ, and set g to be a random product of Artin generators of length λ. So $g = \prod_{i=1}^{\lambda} b_{n_i}^{\epsilon_i}$, where the integers $n_i \in \{1, 2, \ldots, N-1\}$ and $\epsilon_i \in \{-1, 1\}$ are outputs of our pseudorandom number generator. We choose λ large enough so that the elements $1 * g$ are approximately uniform in Ω.)

We then define a function $f : \Omega \to \Omega$ by

$$f(x) = \begin{cases} 1g & \text{if } r(x) = (g, 0), \\ \omega g^{-1} & \text{if } r(x) = (g, 1). \end{cases}$$

Next, we find a collision for f: elements $x_1, x_2 \in \Omega$ with $f(x_1) = f(x_2)$. This can be done using standard collision-finding techniques (such as Floyd cycle finding) in $O(\sqrt{|\Omega|})$ expected time, and constant memory.

Now, $r(x_1) = (g_1, a_1)$ and $r(x_2) = (g_2, a_2)$ for some group elements $g_1, g_2 \in G$ and bits $a_1, a_2 \in \{0, 1\}$. We repeat this process using different pseudorandom generators (and so different functions f) until we have a collision with $a_1 \neq a_2$. The event that $a_1 \neq a_2$ occurs with probability about $1/2$, provided that the distribution of the elements $1g_1 \in \Omega$ and $\omega g_2^{-1} \in \Omega$ are each approximately uniformly distributed in Ω. So the expected number of functions f we need to consider is constant (just 2). So we can find a collision of this restricted form in $O(\sqrt{|\Omega|})$ expected time.

Without loss of generality, suppose $a_1 = 0$ and $a_2 = 1$. By the definition of f, we have $1g_1 = \omega g_2^{-1}$, and so $g = g_1 g_2$ is the element we are seeking.

As a final comment, we note that the following algorithm (which uses $O(|\sqrt{\Omega}|)$ memory) is an alternative to the algorithm above: First, choose $O(\sqrt{|\Omega|})$ elements $g_1 \in G$ at random. For each element, compute and store the pair $(1g_1, g_1)$. Store the pairs in a data structure so that the pair with a given first coordinate (if it exists) can be efficiently retrieved. Secondly, choose elements $g_2 \in G$ at random. For each element, compute ωg_2^{-1}. Repeat until ωg_2^{-1} is equal to the first coordinate of a pair generated earlier. Then we have found g_1 and g_2 such that that $\omega g_2^{-1} = 1g_1$, and so we may return $g = g_1 g_2$.

From: Derek Atkins <datkins@securerf.com>
Sent: Monday, January 22, 2018 12:12 PM
To: pqc-forum@list.nist.gov; pqc-comments
Subject: OFFICIAL COMMENT: WalnutDSA typographic error

Dear All,

We have found a typographic error in the WalnutDSA specification in section 5.1 on page 5. In the document we incorrectly repeated the use of variable, so where the spec says "We assume the hash output M is 4l bits long" this should be "4d bits", meaning we assume that the hash length is a multiple of 4 bits.

We are sorry for the error.

The WalnutDSA Team
--
Derek Atkins
Chief Technology Officer
SecureRF Corporation

Office: 203.227.3151 x1343
Direct: 617.623.3745
Mobile: 617.290.5355
Email: DAtkins@SecureRF.com

From: Derek Atkins <datkins@securerf.com>
Sent: Monday, January 22, 2018 10:41 PM
To: S.Blackburn@rhul.ac.uk; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: WalnutDSA

Dear Simon,

Thank you for pointing this out. A very small parameter increase from N=8 to N=10 will fix this with only a small increase to sizes and performance.

Thanks,

The WalnutDSA Team

On Mon, 2018-01-22 at 10:54 +0000, Blackburn, S wrote:

> Dear All,
>
> There are fewer than 2^512 possibilities for each half of the public key in WalnutDSA. This makes the scheme vulnerable to `square root' attacks (using an approach similar to Pollard-rho). It is recommended that parameter sizes should be increased so this approach is no longer feasible.
>
> More details are given in the attached pdf, which is also available at the following link:
>
> https://www.dropbox.com/s/dyje618qfb4zvsj/Walnut_private_key_2.pdf?dl=0
>
> Yours,
>
> Simon Blackburn
>
> ---
> Simon R. Blackburn
> Professor of Pure Mathematics
> Department of Mathematics
> Royal Holloway University of London
> Egham, Surrey TW20 0EX, United Kingdom
> Tel: (+44) (0)1784 443422
> E-mail: S.Blackburn@rhul.ac.uk
> Web: http://www.ma.rhul.ac.uk/sblackburn

- -
Derek Atkins
Chief Technology Officer
SecureRF Corporation

Office: 203.227.3151 x1343
Direct: 617.623.3745
Mobile: 617.290.5355
Email: DAtkins@SecureRF.com

From: Ward Beullens <ward.beullens@student.kuleuven.be>
Sent: Tuesday, January 23, 2018 11:28 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: [pqc-forum] OFFICIAL COMMENT: WalnutDSA

Dear all,

In order to verify a WalnutDSA signature-message pair (s,m) one first converts the message m into a braid, and then one checks if this braid satisfies some condition involving the signature s.

The way that messages are converted to braids consists of 2 steps:

1) Compute a 512-bit hash digest of the message h. (For the 256-bit security parameters)
2) Convert the hash to a braid with the encoder algorithm described in section 7.

The problem with this approach is that the second step is not injective (e.g 0x0044 , 0x4400 , 0x0404 , 0x0440 are all mapped to the same braid
g_1^6), and that the number of braids that are encodings of messages is significantly less than 2^512. This opens up the possibility of an EUF-CMA attack. An attacker finds two messages m_1 and m_2 that are converted to the same braid, asks the signing algorithm for a valid signature s for m_1, and returns the valid signature-message pair (s,m_2).

Using a recurrence relation I calculated that the number of different encoding of the 2^512 possible hash values is at most 2^483.67, so collision finding takes roughly 2^242 computations, slightly less than the target of 2^256.

The solution to the problem is to use a hash function with a longer output, or to use an encoding function which is injective. I believe both methods will lead to a slight increase to the size of the signatures.

Kind regards,
Ward

From: Derek Atkins <datkins@securerf.com>
Sent: Tuesday, January 23, 2018 11:48 AM
To: ward.beullens@student.kuleuven.be; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: WalnutDSA

Dear Ward, all,

Regarding your first comment, the structure of the security proof we give is modeled on that of "Security Proofs and Signature Schemes" by Pointcheval and Stern (EUROCRYPT 1996). One has to have access to valid signatures for chosen messages. In their paper, the authors give two models for this (cf. Fig 4 in their paper). In the first model the attacker has access to a signer Sigma with a private key who must make queries to a hash function f in the process of producing valid signatures. In the second, the signer Sigma is replaced by a simulator S. The simulator plays the same role; it produces signatures on message inputs, but does not do so by querying the hash function f using the private key. We will add another explicit reference to this paper in ours. The point of the proof is that if one assumes the existence of such a simulator, then one could use it to break the hard problem.

For your second comment, the forger uses the simulator to produce the signatures. This was said explicitly at the beginning of the security proof.

If you need further clarification of the proof we can discuss it further offline.

The WalnutDSA Team

On Thu, 2018-01-18 at 17:28 +0100, Ward Beullens wrote:

> Dear Derek, All,
>
> I agree that WalnutDSA is not broken yet, but I don't think the attack is insignificant either. After all, the WalnutDSA team decided it was worthwhile to modify the original scheme in an attempt to block this attack, even at the cost of doubling the public and private key sizes. (Will this modification be kept, now that is is clear that it does not really block this attack?)
>
> About the security proof: I don't think my concern was addressed. The proof uses a EUF-CMA forger, but does not give a way to answer the signing queries of this forger. You say "we define a forger to be a randomized algorithm that can make hash queries to a random oracle and signature queries to a simulator that does not know Priv(S) but can simulate an honest signer."
>
> This seems very peculiar to me, do you assume the existence of such a simulator? If this assumption were true, then WalnutDSA is not secure.
>
> The security proof mentions "The Forger F is defined to be a randomized algorithm that on input a message m ∈{0, 1}^∗, a signers public key Pub(S), and a coin ρ, outputs a 4-tuple (m, h, g_ρ,s) , where h = H(m) and g_ρ = G_ρ(V) and V ← Cloak, s ← DC_m,V,H,G. It is assumed that the probability that (m, h, gρ,s) is a valid WalnutDSA-I signature is non-negligible."
>
> This looks like a universal (the message can be freely chosen), key-only attack (there is no mention of signing queries). How does this relate to the claim that Walnut is EUF-CMA secure?
>
> Kind regards,
> Ward

From: Derek Atkins <datkins@securerf.com>
Sent: Tuesday, January 23, 2018 11:50 AM
To: ward.beullens@student.kuleuven.be; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: WalnutDSA

Dear Ward, All,

Thank you for bringing this to our attention. The encoder can easily be fixed by changing to a 2-bit encoder where each 2-bit input maps directly to a generator g_i. This minor change would make the encoder properly injective.

Thanks,

The WalnutDSA Team

On Tue, 2018-01-23 at 17:28 +0100, Ward Beullens wrote:

> Dear all,
>
> In order to verify a WalnutDSA signature-message pair (s,m) one first converts the message m into a braid, and then one checks if this braid satisfies some condition involving the signature s.
>
> The way that messages are converted to braids consists of 2 steps:
>
> 1) Compute a 512-bit hash digest of the message h. (For the 256-bit security parameters)
> 2) Convert the hash to a braid with the encoder algorithm described in section 7.
>
> The problem with this approach is that the second step is not injective (e.g 0x0044 , 0x4400 , 0x0404 , 0x0440 are all mapped to the same braid g_1^6), and that the number of braids that are encodings of messages is significantly less than 2^512. This opens up the possibility of an EUF-CMA attack. An attacker finds two messages m_1 and m_2 that are converted to the same braid, asks the signing algorithm for a valid signature s for m_1, and returns the valid signature-message pair (s,m_2).
>
> Using a recurrence relation I calculated that the number of different encoding of the 2^512 possible hash values is at most 2^483.67, so collision finding takes roughly 2^242 computations, slightly less than the target of 2^256.
>
> The solution to the problem is to use a hash function with a longer output, or to use an encoding function which is injective. I believe both methods will lead to a slight increase to the size of the signatures.
>
> Kind regards,
> Ward

--
Derek Atkins
Chief Technology Officer
SecureRF Corporation

Office: 203.227.3151 x1343

From: Ward Beullens <ward.beullens@student.kuleuven.be>
Sent: Tuesday, January 23, 2018 4:15 PM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: WalnutDSA

Dear WalnutDSA team,

I had a look at the paper you reference. An entire page of the security proof is devoted to showing how the simulator S is constructed (i.e. the proof of Lemma 8). This part is missing from the WalnutDSA security proof and this is basically the concern that I raised.

There is a different way to see that something is wrong with the security proof. Consider these 2 parameters: the length of the output of the hash function that is used and the length limit that has to be imposed on the signatures. If these parameters are chosen poorly, the scheme is not secure, so in that case the security proof should fail somewhere. However, these 2 parameters are not mentioned in the security proof, nor in the formulation of the hard problem that EUF-CMA security is supposedly reduced to.

If the security proof is correct, can you please pinpoint where the security proof fails if
1) The length of the hash function that is used is too small, and
2) The verification algorithm does not impose a length limit on the signatures.

Kind regards,
Ward

Op 23/01/2018 om 17:47 schreef Derek Atkins:

> Dear Ward, all,
>
> Regarding your first comment, the structure of the security proof we give is modeled on that of "Security Proofs and Signature Schemes" by Pointcheval and Stern (EUROCRYPT 1996). One has to have access to valid signatures for chosen messages. In their paper, the authors give two models for this (cf. Fig 4 in their paper). In the first model the attacker has access to a signer Sigma with a private key who must make queries to a hash function f in the process of producing valid signatures. In the second, the signer Sigma is replaced by a simulator S. The simulator plays the same role; it produces signatures on message inputs, but does not do so by querying the hash function f using the private key. We will add another explicit reference to this paper in ours. The point of the proof is that if one assumes the existence of such a simulator, then one could use it to break the hard problem.
>
> For your second comment, the forger uses the simulator to produce the signatures. This was said explicitly at the beginning of the security proof.
>
> If you need further clarification of the proof we can discuss it further offline.
>
> The WalnutDSA Team
>
> On Thu, 2018-01-18 at 17:28 +0100, Ward Beullens wrote:
>
> > Dear Derek, All,
> >
> > I agree that WalnutDSA is not broken yet, but I don't think the attack is insignificant either. After all, the WalnutDSA team decided it was worthwhile to modify the original

From: Ward Beullens <ward@beullens.com>
Sent: Thursday, February 01, 2018 10:21 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: WalnutDSA
Attachments: WalnutDSAScript.sage.sage

Dear All,

There is an attack that breaks EUF-CMA security of the walnutDSA scheme with 2^56 operations for the 256 security bit parameters, and 2^35 operations for the 128 bit security level. I have communicated with the designers, they have confirmed that the attack seems to work, and they proposed countermeasures.

My previous attack exploited the fact that the range of E composed with H is smaller than 2^512, in order to do a collision attack. Now I have found that this range, after the composition with P gets much smaller.
So, similar to my previous attack, one can easily find two messages m_1 and m_2 such that P(E(H(m_1))) = P(E(H(m_2))), breaking EUF-CMA security.

Using sage, (I attached the script) I picked a sample of random hashes h_i and computed the matrix part of P(E(h_i)) for these hashes. Then I calculated that these matrices span a subspace of dimension only 14.
This means that P(E(H(m))) can reach at most (2^8)^14 = 2^112 values, so a collision search costs roughly 2^56 steps, much less than the security level of 2^256 that was aimed for.

For the 128 bit security parameters the collision finding would require roughly sqrt((2^5)^14) = 2^35 E-multiplications, so it should be feasible to demonstrate this attack in practice.

Kind regards,
Ward

```
import random
import itertools

N = 8;
B  = BraidGroup(N,'b');
K = GF(2^8);
MS = MatrixSpace(K,N,N);

B.inject_variables();

S = Set();

def RandomNonzeroElement():
    while True:
        r = K.random_element()
        if r != 0:
            return r;

def PBG(i,j):
    return B([a for a in range(j-1,i,-1) ] + [ i , i ] + [ -a for a in range(i+1,j)
]);

# Choosing T values
Tvals = [RandomNonzeroElement() for i in range(0,N)];
Tvals[0] = K.one()
Tvals[1] = K.one()

CBOne = (MS.identity_matrix(),Permutations(N).identity())

#One step of E-multiplication
def EmulStep(x,g):
    M,p = x;
    M = copy(M);
    p = copy(p);

    if (g>0) :
        a = Tvals[p(g)-1];
        b = -a;
        c = 1;
    else :
        a = 1;
        b = - Tvals[p(-g+1)-1]^(-1);
        c = -b;

    g = abs(g);

    if(g>1):
        M.set_column(g-2 , M.column(g-2) + a* M.column(g-1));

    M.set_column(g , M.column(g) + c*M.column(g-1));
    M.set_column(g-1 , b*M.column(g-1));
    return (M,B([g]).permutation()*p);

#E-multiplication
def Emul(x,b):
    return reduce(EmulStep,[x]+list(b.Tietze()));

#The number of generators per hash value
Length = 256;
#The number of hash values
Samples = 40;

#The generators of the free group
```

```
G = [ PBG(1,N), PBG(3,N) , PBG(5,N) , PBG(7,N) ]

BigMatrix = matrix(K,N*N,Samples);

for i in range(0,Samples):
    x = CBOne;
    for j in range(0,Length):
        x = Emul(x,random.choice(G));
    m,p = x
    BigMatrix.set_column(i,m.list())
    print("progress = "+str(100.0*(i)/Samples)+"%")


print("Dimension of span of P(E(h_i)) is :")
print(BigMatrix.rank())
```

From: Derek Atkins <datkins@securerf.com>
Sent: Friday, February 02, 2018 1:33 PM
To: ward@beullens.com; pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: WalnutDSA
Attachments: GeneralWalnutEncoder.sage

Dear Ward, All,

Thank you for your comment. You already had pointed out an issue with the encoder that we agreed with and that we would address with one of several available options. Moreover, we acknowledged this related comment which points out that the "still-unchanged" encoding as given in the specification resulted in a too-small dimensional (14) subspace generated by the images of the public keys. As we noted in our most recent response to you, the introduction of an alternate encoder (see following suggestion) will address both of these issues.

There any many different ways to injectively encode messages into the free subgroup of the pure braid group, and we can easily opt for a method that takes this into account. One possible way that is similar to what we already do is to break the hashed message into 128 (or 256) 2-bit blocks and to choose a fixed finite ordering S of certain 4-tuples of generators. Then as one traverses the blocks one selects a generator from the current tuple from S; at each successive block one takes the next tuple from S, and then cycles back to the beginning when the end of S is reached. More precisely, suppose we work in B12. One could take S to be the list

[579B,468A,3579,2468,1357,2468,3579,468A]

and if the hashed message has 2-bit blocks

BLK1, BLK2, ... BLK128

then one does the following:

use BLK1 to select from g5, g7, g9, gB
use BLK2 to select from g4, g6, g8, gA
...
use BLK5 to select from g1, g3, g5, g7
use BLK6 to select from g2, g4, g6, g8
…
use BLK8 to select from g4, g6, g8, gA
use BLK9 to select from g5, g7, g9, gB
...
and so on. It is easy to verify that one obtains a large subspace this way (see attached Sage program). For example, in B12 one finds a 102-dimensional subspace generated by the public keys. Over F256 this subspace contains 2^816 elements, which should be more than sufficient. Moreover, having the entries repeat in this manner not only remains injective, continues to hold a dimension of 102 in B12, but also results in a shorter encoding which means a shorter signature.

The WalnutDSA Team

On Thu, 2018-02-01 at 16:21 +0100, Ward Beullens wrote:

> Dear All,
>
> There is an attack that breaks EUF-CMA security of the walnutDSA scheme

```
##  Compute the dimentionality of the WalnutDSA Encoder Matrix

import random
import itertools

# Note;  If you change N, you should change the definitions of G
#        down near the end.
N = 12;
B  = BraidGroup(N,'b');
K = GF(2^8);
MS = MatrixSpace(K,N,N);

B.inject_variables();

S = Set();

def RandomNonzeroElement():
    while True:
        r = K.random_element()
        if r != 0:
            return r;

def myPerm(b):
    p = b.permutation();
    return p*Permutations(4).identity();

def isPure(a):
    #print(a,a.permutation().cycle_string());
    return a.permutation().cycle_string() == "()";

def getAllBraids(l):
    def isReduced(a):
        for i in range(0,len(a)-1):
            if a[i] == -a[i+1]:
                return False;
        return True;

    def toBraid(a):
        return B(a);

    def AllBraids(l):
        S = [i for i in range(1,N)] + [-i for i in range(1,N)];
        return
itertools.imap(toBraid,itertools.ifilter(isReduced,itertools.product(*([S]*l))));

    S = set();
    for i in range(0,l+1):
        S.update(AllBraids(i))
    return S;

def getAllPureBraids(l):
    l = l/2;
    D = dict()
    for b in getAllBraids(l):
        p = myPerm(b);
        if p in D:
            D[p].add(b);
        else:
            D[p] = set([b]);

    PB = set();
    for p in D:
        pinv = p.inverse();
```

```
        for a in D[p]:
            for b in D[pinv]:
                PB.add(a*b);
                PB.add(b*a);
    return PB;

def PBG(i,j):
    return B([a for a in range(j-1,i,-1) ] + [ i , i ] + [ -a for a in range(i+1,j)
]);

def getPureBraidGenerators():
    S = set();
    for i in range(1,N):
        for j in range(i+1,N):
            S.add(PBG(i,j))
    return S;

# Choosing T values
Tvals = [RandomNonzeroElement() for i in range(0,N)];
Tvals[0] = K.one()
Tvals[1] = K.one()

CBOne = (MS.identity_matrix(),Permutations(N).identity())

def EmulStep(x,g):
    M,p = x;
    M = copy(M);
    p = copy(p);

    if (g>0) :
        a = Tvals[p(g)-1];
        b = -a;
        c = 1;
    else :
        a = 1;
        b = - Tvals[p(-g+1)-1]^(-1);
        c = -b;

    g = abs(g);

    if(g>1):
        M.set_column(g-2 , M.column(g-2) + a* M.column(g-1));

    M.set_column(g , M.column(g) + c*M.column(g-1));
    M.set_column(g-1 , b*M.column(g-1));
    return (M,B([g]).permutation()*p);

def Emul(x,b):
    return reduce(EmulStep,[x]+list(b.Tietze()));

#The number of generators per hash value
Length = 256;  #512/2
#The number of hash values (this must be greater than the expected dimension)
Samples = 120;

#The generators of the free group
#  For B12: 579B,468A,3579,2468,1357
G = [ [PBG(5,N), PBG(7,N), PBG(9,N), PBG(11,N)],
      [PBG(4,N), PBG(6,N), PBG(8,N), PBG(10,N)],
      [PBG(3,N), PBG(5,N), PBG(7,N), PBG(9,N)],
      [PBG(2,N), PBG(4,N), PBG(6,N), PBG(8,N)],
      [PBG(1,N), PBG(3,N), PBG(5,N), PBG(7,N)],
      [PBG(2,N), PBG(4,N), PBG(6,N), PBG(8,N)],
```

```
      [PBG(3,N), PBG(5,N), PBG(7,N), PBG(9,N)],
      [PBG(4,N), PBG(6,N), PBG(8,N), PBG(10,N)] ]

BigMatrix = matrix(K,N*N,Samples);

#for i in range(0,10):
#    print("Choosing: " + str(random.choice(G[i%len(G)])))

for i in range(0,Samples):
    x = CBOne;
    for j in range(0,Length):
        x = Emul(x,random.choice(G[j%len(G)]));
    m,p = x
    BigMatrix.set_column(i,m.list())
    print("progress = "+str(100.0*(i)/Samples)+"%")


print("Dimension of span of P(E(h_i)) is :")
print(BigMatrix.rank())
```